XDef 2019

对话 · 交流 · 合作　　前沿 · 实用 · 人才

第八届全国网络与信息安全防护峰会

二进制代码比对分析云平台BigCodeDiff

唐　勇

国防科技大学计算机学院

About us

软件安全智能并行分析
湖南省重点实验室
Intelligent and Parallel Analysis of Software Security lab



用超级计算机算安全

依靠程序分析理论、编译技术、网络攻防、并行计算、智能计算等技术，利用国家超算中心的计算资源，紧贴国家网络空间安全战略需求、国产软件开发测试产业发展需要，开展软件安全方向的前沿技术研究和人才培养

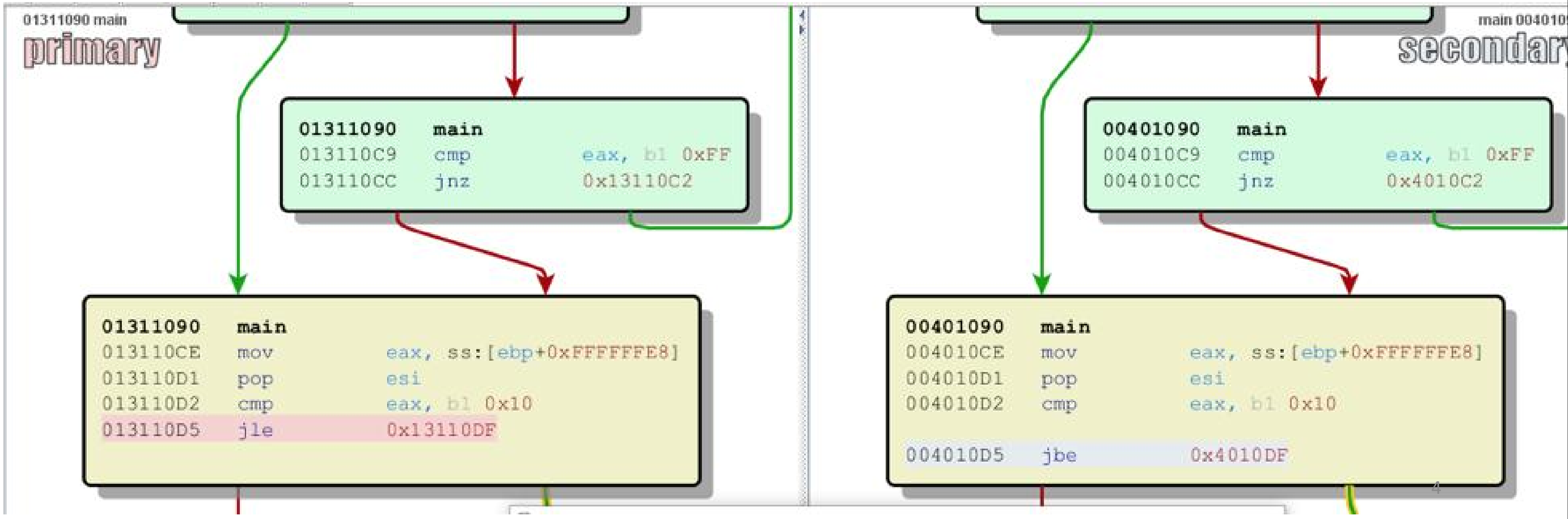
- 软件安全基础理论方法
  - 软件分析基础理论
  - 安全形式化验证
  - 软件内生安全技术
  - 并发程序安全分析
- 超算+软件安全”方向
  - 超算+漏洞挖掘
  - 超算+软件基因分析
  - 超算+密码安全
- 软件智能测试方向
  - 智能单元测试
  - 智能模糊测试
  - 面向领域的智能测试
- 软件供应链安全方向
  - 软件安全开发方法及工具
  - 自主可控信息系统安全评估
  - 软件弹性安全
  - 恶意代码检测与追踪溯源

提 纲

- 二进制代码比对分析的作用
- 研究现状与我们的研究
- 二进制代码分析云平台BigCodeDiff

二进制代码同源比对分析

- 在闭源情况下找到二进制代码之间存在（源）代码复用、存在的差异



二进制比对的应用场景

- 软件剽窃鉴定：随着开源项目的快速增长，软件剽窃已成为维护软件行业的健康可信环境的严重威胁。
- 补丁分析：对更新前后的二进制程序进行相似性分析，得到所更改的内容并分析漏洞成因
- 恶意代码同源检测：通过代码片段的相似性发现恶意代码的家族关系，尤其对APT样本进行细粒度的同源分析是攻击溯源的重要手段
- 逆向分析辅助：通过相似性对比，逆向人员能够P的分析结果应用到它的后续版本P’上，从而减轻逆向工作量
- 通过已知漏洞识别发现未知漏洞
- 自主可控分析：自主可控真假难辨
- 代码成分分析
- ……

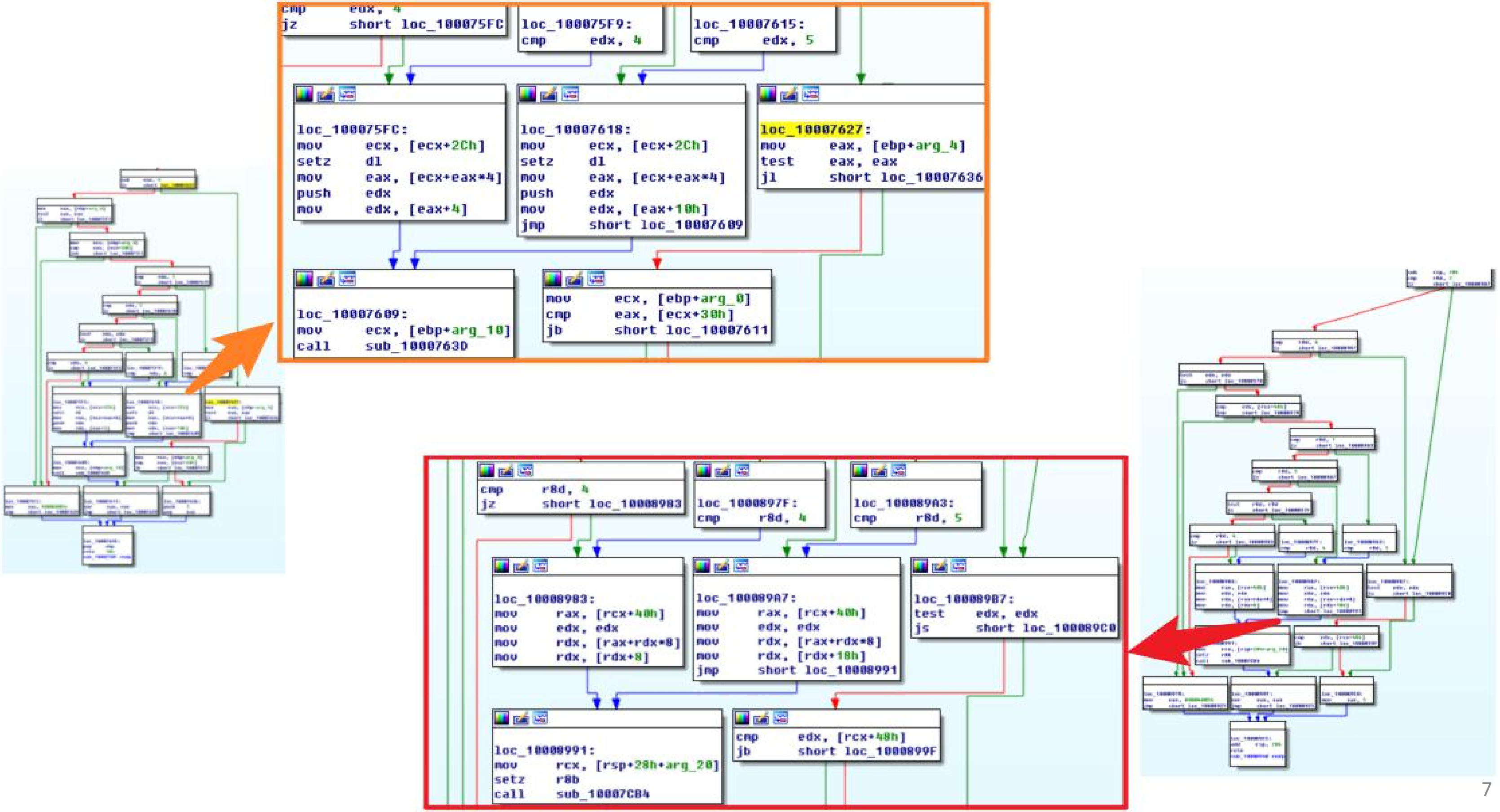
二进制代码比对分析作用

- 软件成分分析与剽窃检测

通过对可疑软件的二进制与受保护软件的二进制进行比对分析，能够检测可疑软件中代码的抄袭情况。

- 例子

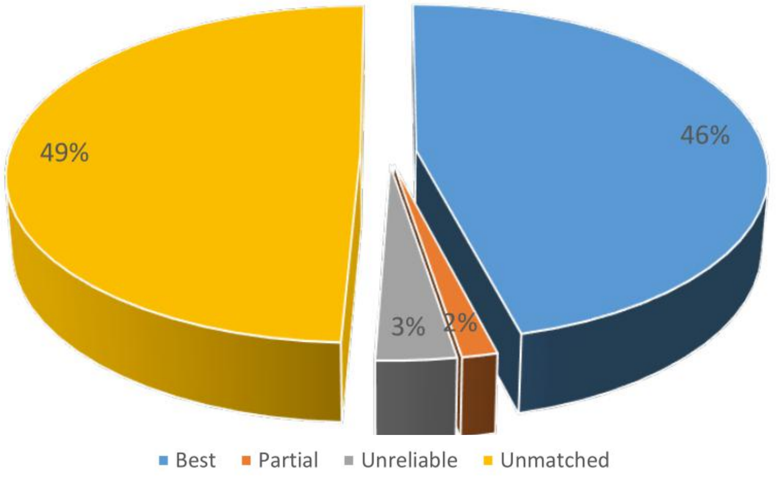
我们通过测试多款市面上的压缩软件与开源软件7zip比对，发现了几款软件存在大量函数和代码片断与7zip相同或逻辑相似。


jz short loc_100075FC
loc_100075F9:
cmp edx, 4
loc_10007615:
cmp edx, 5
loc_100075FC:
mov ecx, [ecx+2Ch]
setz dl
mov eax, [ecx+eax*4]
push edx
mov edx, [eax+4]
loc_10007618:
mov ecx, [ecx+2Ch]
setz dl
mov eax, [ecx+eax*4]
push edx
mov edx, [eax+10h]
jmp short loc_10007609
loc_10007627:
mov eax, [ebp+arg_4]
test eax, eax
jl short loc_10007636
loc_10007609:
mov ecx, [ebp+arg_10]
call sub_1000763D
mov ecx, [ebp+arg_0]
cmp eax, [ecx+30h]
jb short loc_10007611
cmp r8d, 4
jz short loc_10008983
loc_1000897F:
cmp r8d, 4
loc_100089A3:
cmp r8d, 5
loc_10008983:
mov rax, [rcx+40h]
mov edx, edx
mov rdx, [rax+rdx*8]
mov rdx, [rdx+8]
loc_100089A7:
mov rax, [rcx+40h]
mov edx, edx
mov rdx, [rax+rdx*8]
mov rdx, [rdx+18h]
jmp short loc_10008991
loc_100089B7:
test edx, edx
js short loc_100089C0
loc_10008991:
mov rcx, [rsp+28h+arg_20]
setz r8b
call sub_10007CB4
cmp edx, [rcx+48h]
jb short loc_1000899F

- 例子

某厂商交换机和思科Catalyst 2960X，从公开渠道获取的固件进行比对分析。发现存在高度相似性，可以断定两份固件大部分代码同源

对象 / 参数	样本	参考
名称	s5960-universalk9-mz.122-5.I2	c2960x-universalk9-mz.152-5.E1
发布时间	未知	2016 年 11 月 16 日
文件大小（byte）	25760768	25774080
代码长度（byte）	65363360	65371340
数据长度（byte）	65213044	65247092
函数数量	255690	255681



对二者的相似性分析采取了最基本、最保守的对比算法，得出其中46%的函数完全相同，5%存在不同程度的差异

二进制代码比对分析作用

- 已知漏洞识别

  通过对待检测二进制代码与漏洞库中存在漏洞的二进制函数进行比对，定位漏洞的影响范围。

- 例子

  CVE-2019-1208漏洞是微软今年6月公布的一个vbscript漏洞，随后在微软的9月补丁星期二发表并修复，报告中提到这个漏洞通过补丁比可以被发现在rtJoin函数中，在对数组内的元素进行操作前后，加了一对SafeArrayLock/SafeArrayUnlock函数。


primary
secondary
00036EBD rtJoin
00036F2F push ds:[edi] // psa
00036F31 call ds:[__imp__SafeArrayLock@4] //
__imp__SafeArrayLock@4
00036F37 mov ss:[ebp+var_C], eax
00036F3A test eax, eax

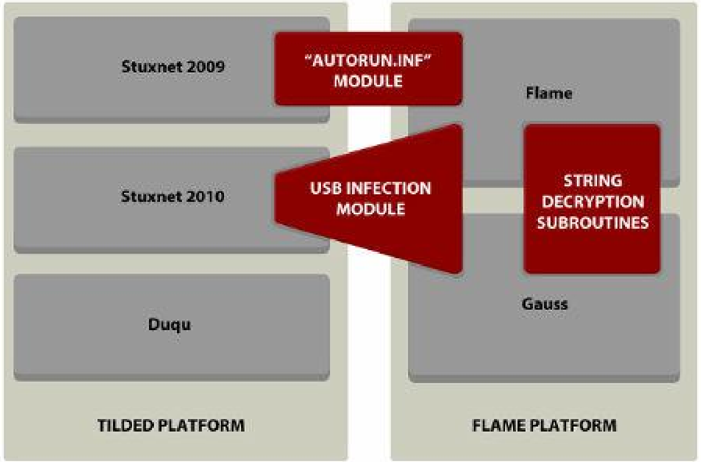
二进制代码比对分析作用

- 恶意代码同源分析

通过代码片段的相似性发现恶意代码的家族关系，尤其对APT样本进行细粒度的同源分析是攻击溯源的重要手段。

例子：从震网/火焰/毒区/高斯病毒样本进行APT溯源分析

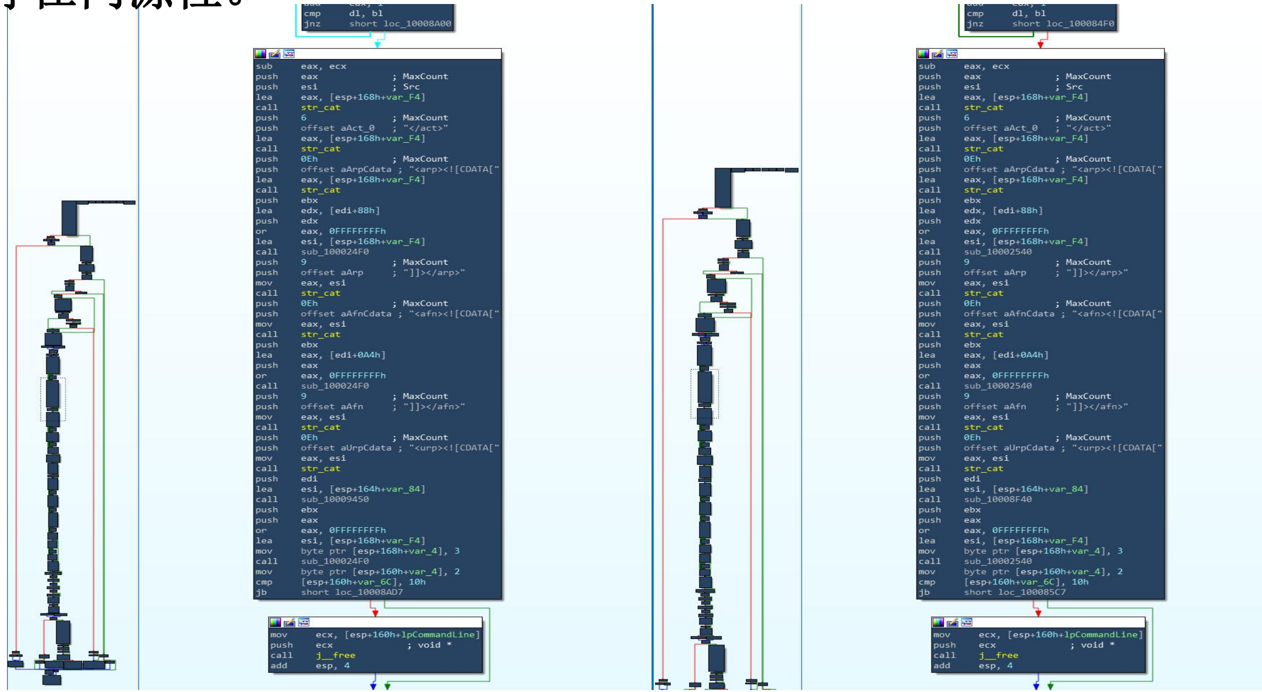
The relationship of Stuxnet, Duqu, Flame and Gauss



© 2012 Kaspersky Lab ZAO. All Rights Reserved.

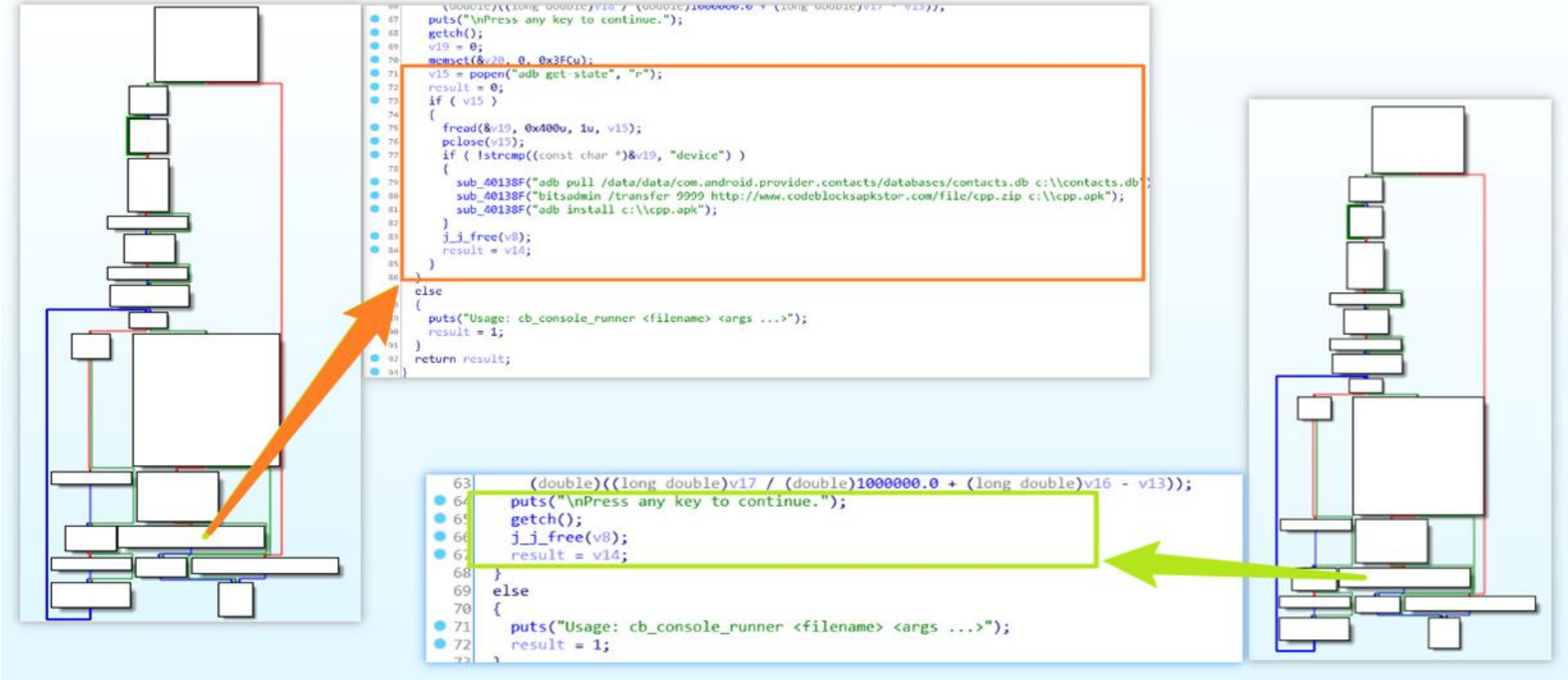
- 例子

  不久前多款后门病毒利用某驱动软件短时间内大规模感染数万台电脑，。通过二进制同源代码比对，我们发现推送病毒执行的升级模块，与“人生日历”升级模块代码存在同源性。



• 例子二

在2018年的阿里“功守道”软件供应链安全大赛中，国防科大团队以“holiday”战队代表参赛。利用大代码比对台快速的找出了大量同源差异性函数，在这些函数中快速的定位出了隐藏的恶意代码，凭借这一优势在二进制PE分站赛中夺得第一名。



二进制代码比对分析作用

- 逆向工程

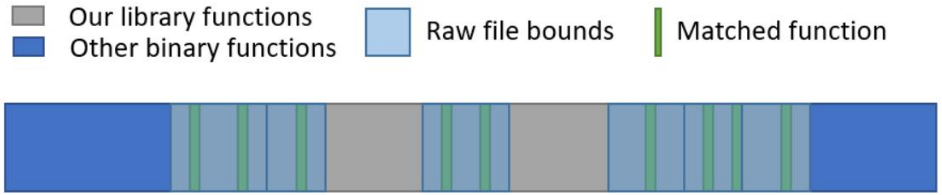
通过相似性对比，逆向人员能够P的分析结果应用到它的后续版本P'上，从而减轻逆向工作量

- 例子

在对某IOT设备逆向过程中，二进制代码比能够直接帮助逆向人员完成了前期的大量工作。通过代码溯源可以告诉逆向人员固件中使用的开源库情况，通过插件配置甚至能够标注出部分模块函数功能，直接搜索易受攻击的函数。

对某IOT设备固件溯源分析识别情况:

- libpng: 1.2.29
- zlib: 1.2.3
- OpenSSL: 1.0.1.j
- mDNSResponder: unknown
- gSOAP: 2.7



	libpng	zlib	OpenSSL*
# functions	300	75	3514
# matched functions (TP)	277	68	2857
# unmatched referenced functions	0	3	454
# mismatched functions (FP)	0	0	8
Percentage matched	92% (100%)	91% (96%)	82% (85%)

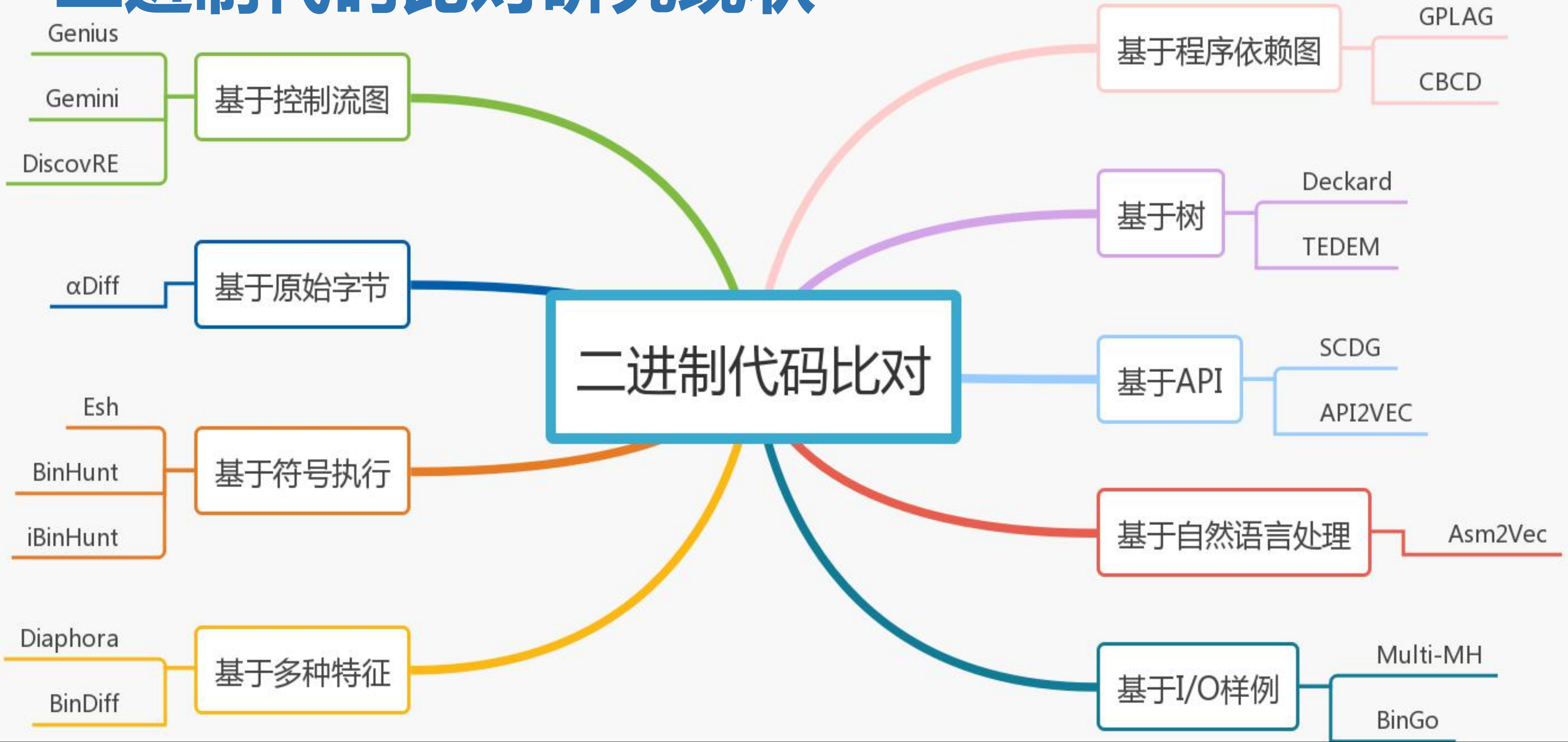
提 纲

- 二进制代码比对分析的作用
- 研究现状与我们的研究
- 二进制代码分析云平台BigCodeDiff

二进制同源分析的技术挑战

- 粒度：识别同源代码的粒度（片段、函数、或整个二进制文件），独立越细难度越大
- 准确性：高准确性可以大大减少后续人工处理开销，应用结果可信
- 跨平台：相同源代码可能被编译成不同平台（指令集、操作系统）上的二进制代码，需要不受平台特性影响识别代码同源
- 抗混淆：二进制代码被混淆变换，加大同源分析的难度，要从语义层面识别同源代码
- 性能：对于大代码，海量代码的细粒度比对存在性能问题，需要并行、可扩展的解决方案

二进制代码比对研究现状



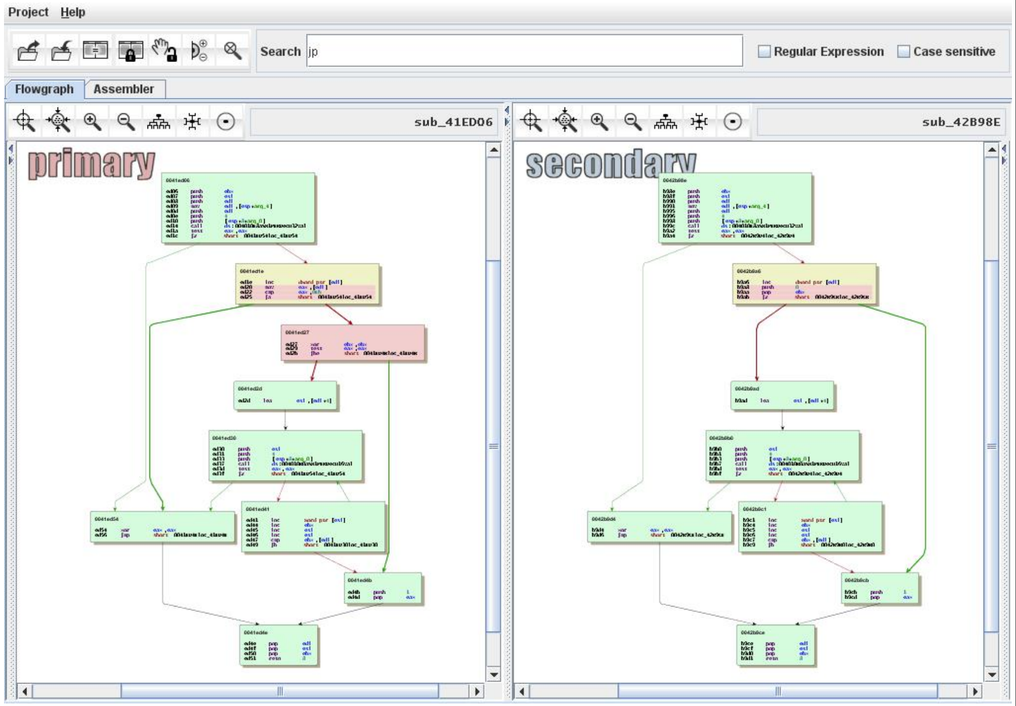
二进制代码比对研究现状

- BinDiff [Google]:

可跨平台

目标：实现多平台的二进制代码比对

方法：使用多种特征进行同源分析，包括：基本块数量，基本块反编译代码，控制流图，函数调用等等。



二进制代码比对研究现状

- Multi-MH [S&P’ 15]

可跨平台

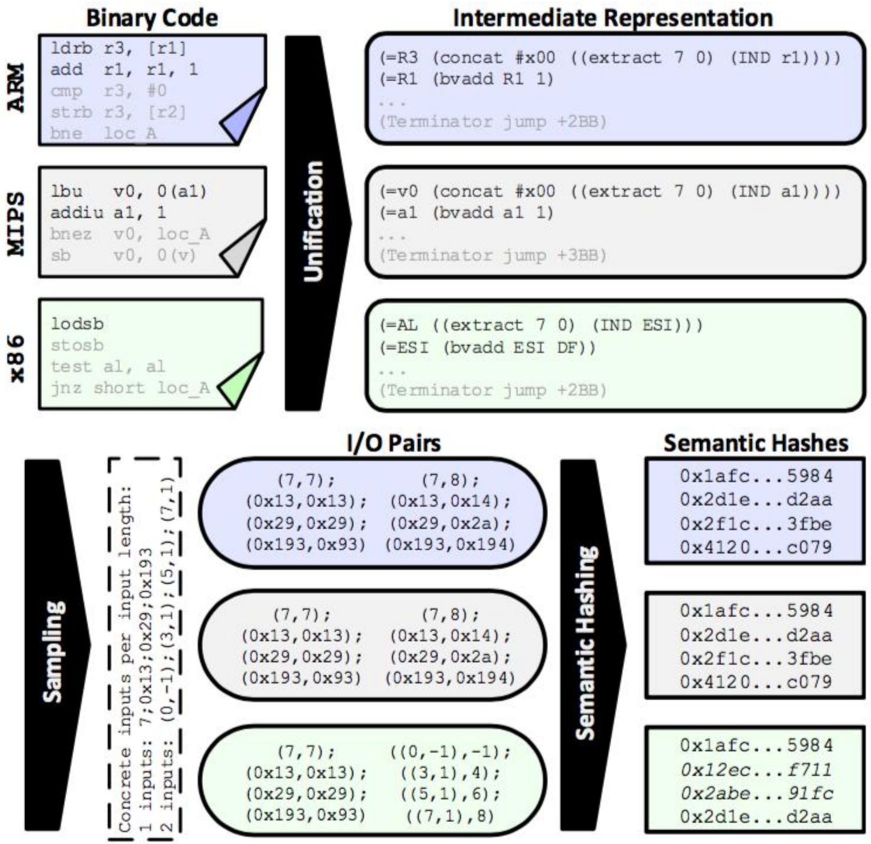
目标：通过求解不同输入的输出结果实现无源代码跨平台二进制比对

方法：1、汇编指令转换成中间表示

2、使用符号执行求解基本块输入/输出值

3、使用MinHash作为LSH求解距离

4、基于CFG(控制流图)匹配实现函数比对



二进制代码比对研究现状

- Multi-MH [S&P’15]

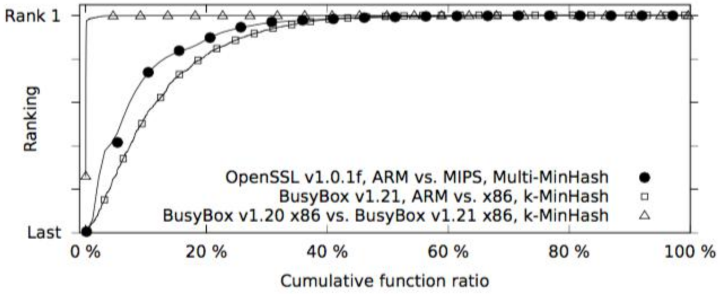
可跨平台

目标：通过求解不同输入的输出结果实现无源代码跨平台二进制比对

实验：采用BusyBox和OpenSSL的相同源码不同平台检测；检测跨平台的Heartbleed漏洞。

意义：为跨平台二进制比对提供一种思路

不足：（1）时间开销过大;(2) 准确率不尽人意;（3）跨编译器和编译优化选项鲁棒性较差



	Multi-MH		Multi-k-MH	
From → To	TLS	DTLS	TLS	DTLS
ARM → MIPS	1;2	1;2	**1;2**	1;2
ARM → x86	1;2	1;2	1;2	1;2
ARM → DD-WRT	1;2	1;2	1;2	1;2
ARM → ReadyNAS	1;2	1;2	1;2	1;2
MIPS → ARM	2;3	3;4	1;2	1;2
MIPS → x86	1;4	1;3	1;2	1;3
MIPS → DD-WRT	1;2	1;2	1;2	1;2
MIPS → ReadyNAS	2;4	6;16	1;2	1;4
x86 → ARM	1;2	1;2	1;2	1;2
x86 → MIPS	1;7	11;21	1;2	1;6
x86 → DD-WRT	70;78	1;2	5;33	1;2
x86 → ReadyNAS	1;2	1;2	1;2	1;2

Heartbleed漏洞检测

二进制代码比对研究现状

- Genius [CCS’16]

可跨平台

目标：基于ACFG检测跨平台的二进制代码相似检测

方法：1、提取函数ACFG

ACFG，属性控制流图，包含基本块特征的控制流图

2、对海量ACFG聚类

3、使用NN算法计算函数向量

4、LSH全局搜索

Type	Feature Name	Weight (α)
Statistical Features	String Constants	10.82
	Numeric Constants	14.47
	No. of Transfer Instructions	6.54
	No. of Calls	66.22
	No. of Instructions	41.37
	No. of Arithmetic Instructions	55.65
Structural Features	No. of offspring	198.67
	Betweeness	30.66

二进制代码比对研究现状

- Genius [CCS’16]

可跨平台

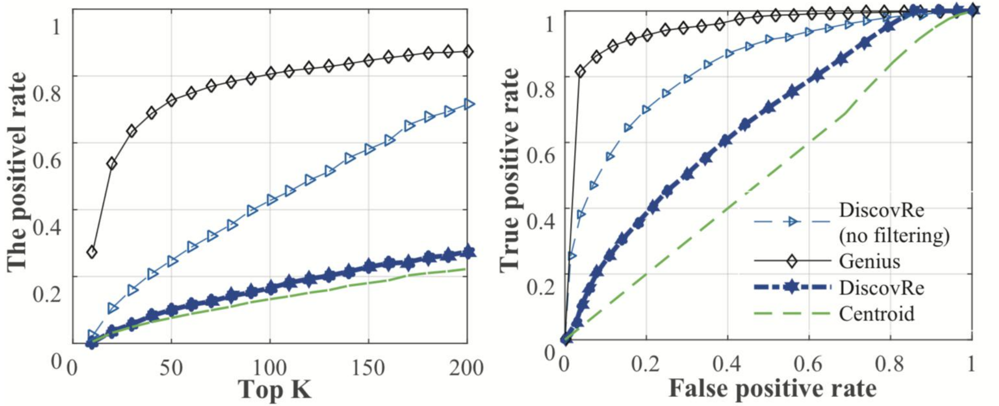
目标：基于ACFG检测跨平台的二进制代码相似检测

实验：发现设备多个已知未修复漏洞，与DiscovRe比较

意义：提出将函数之间图的相似性转换成与中心节点距离的相似性

不足：（1）仅依赖于CFG，无法抵抗改变CFG的代码混淆;(2)人为的选择基本块属性，存在一定局限性

DIR-810L_REVB_FIRMWARE_2.03B02			DIR-810L_REVB_FIRMWARE_2.02.B01		
CVE	Patched	Vulnerability Type	CVE	Patched	Vulnerability Type
CVE-2016-0703	No	Allows man-in-the-middle attack	CVE-2015-0206	No	Memory consumption
CVE-2015-1790	No	NULL pointer dereference	CVE-2014-0160	Yes	Heartbleed
CVE-2015-1791	Yes	Double free	CVE-2015-0289	No	NULL pointer dereference
CVE-2015-0289	No	NULL pointer dereference	CVE-2016-0797	No	Heap memory corruption
CVE-2014-8275	No	Missing sanitation check	CVE-2016-0798	No	Memory consumption
CVE-2015-0209	No	Use-after-free	CVE-2014-3513	No	Memory consumption
CVE-2015-3195	No	Mishandles errors	CVE-2014-3508	No	Information leakage
#	#	#	CVE-2015-0206	No	Memory consumption
#	#	#	CVE-2014-8275	No	Missing sanitation check



a) Recall rates across different threshold K   b) ROC curves for different approaches

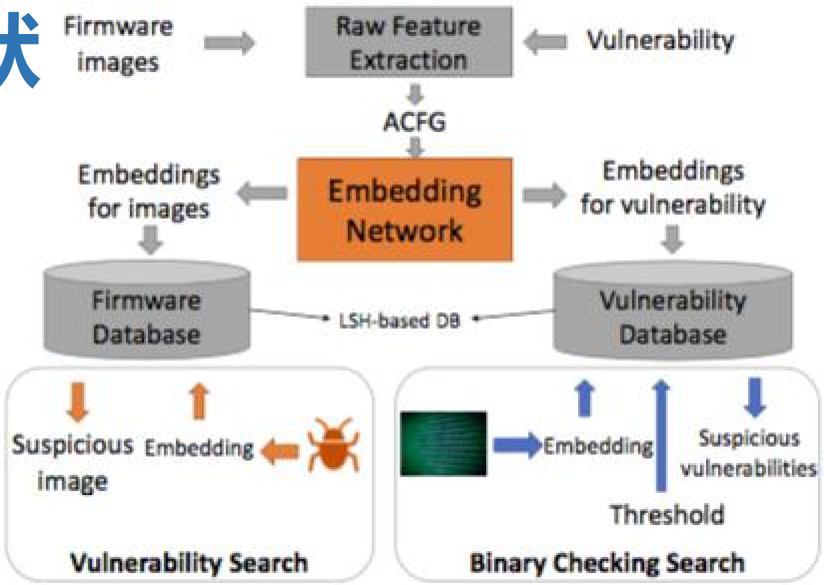
二进制代码比对研究现状

- Gemini [CCS’ 17]

可跨平台

目标：基于ACFG生成向量表示检测跨平台的二进制代码相似检测

方法：1、提取函数ACFG
　　　2、基于神经网络生成向量表示
　　　3、使用孪生网络实现相似性检测



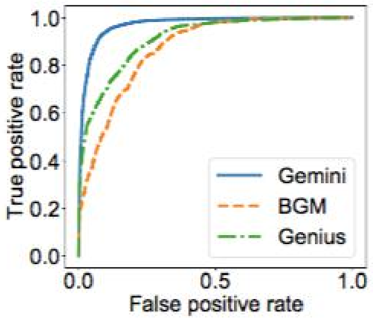
Type	Attribute name
Block-level attributes	String Constants Numeric Constants No. of Transfer Instructions No. of Calls No. of Instructions No. of Arithmetic Instructions
Inter-block attributes	No. of offspring Betweenness

二进制代码比对研究现状

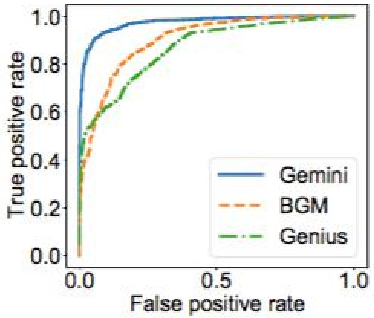
- Gemini [CCS’17]

可跨平台

目标：基于ACFG生成向量表示检测跨平台的二进制代码相似检测



(a) Results on the similarity testing set

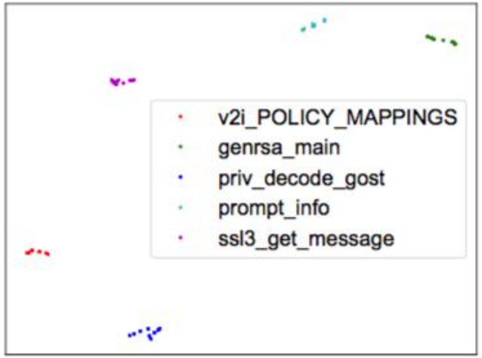


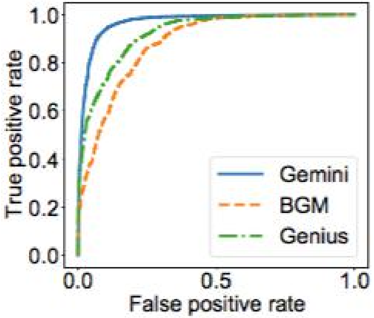
(b) Results on the large-graph subset

实验：与Genius比较

意义：提出使用孪生网络实现相似性比较

不足：（1）仅依赖于CFG，无法抵抗改变CFG的代码混淆;(2)人为的选择基本块属性，存在一定局限性





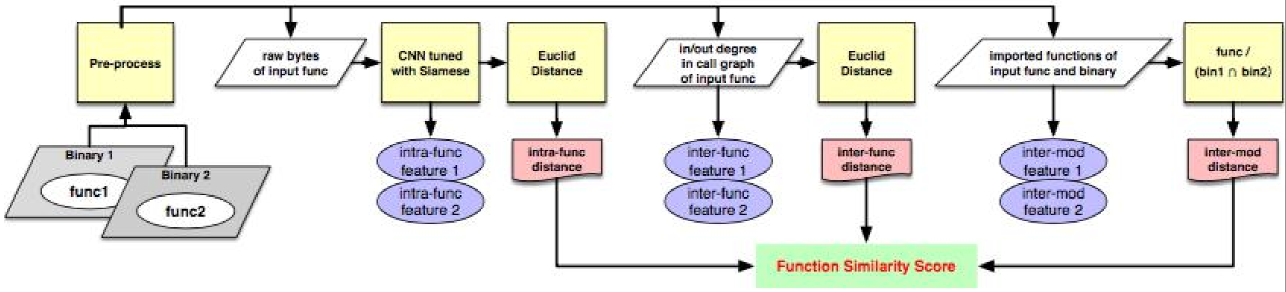
(c) Results on the small-graph subset

二进制代码比对研究现状

- αDiff [ASE’18]

可跨平台

目标：对跨版本，跨编译器，跨平台的二进制进行相似代码检测



特征提取：原始字节，函数调用图，导入函数

方法：1、基于CNN(卷积神经网络)孪生网络将原始字节转成向量 $D1(I_q, I_t) = \|f(I_q;\theta) - f(I_t;\theta)\|$

2、调用图的出度入度 $g(I_q) = (in(I_q), out(I_q))$ $D2(I_q, I_t) = \|g(I_q) - g(I_t)\|$

3、导入函数的匹配

$$D3(I_q, I_t) = \|h(imp(I_q), imp(B_q) \cap imp(B_t)) - h(imp(I_t), imp(B_q) \cap imp(B_t))\|$$

二进制代码比对研究现状

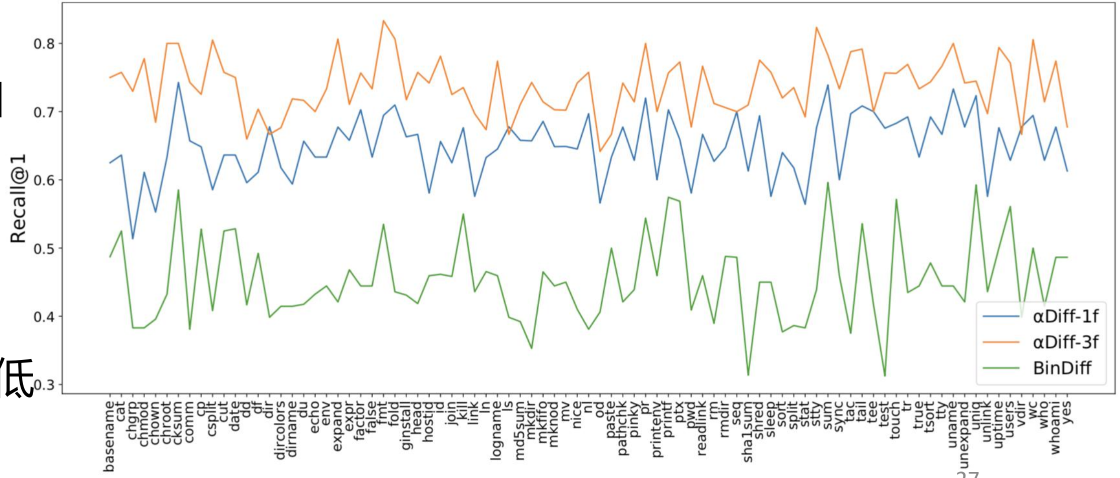
- αDiff [ASE'18]

可跨平台

目标：对跨版本，跨编译器，跨平台的二进制进行相似代码检测

vulnerability	alias	binaries		BinDiff		αDiff		
		pre	post	vul-func found?	Recall@1	vul-func found in top-1?	Recall@1	MRR
CVE-2014-0160	Heartbleed	openssl-1.0.1f	openssl-1.0.1g	×	0.371	√	0.609	0.695
CVE-2014-6271	Shellshock	bash-4.3	bash-4.3.30	×	0.485	√	0.559	0.577
CVE-2014-4877		wget-1.15	wget-1.16	×	0.555	√	0.691	0.664
CVE-2014-7169	Shellshock2	bash-4.3	bash-4.3.30	×	0.485	×	0.559	0.577
CVE-2014-9295	Clobberin Time	ntpd-4.27p10	ntpd-4.28	×	0.434	√	0.422	0.364
CVE-2015-3456	Venom	qemu-2.30	qemu-2.40	×	0.276	×	0.301	0.261

实验：（1）与BinDiff比较发现已知漏洞（2）对coreutils进行测试



不足：1、基于先验知识规定权重

2、跨平台TOP-1准确率较低

TOP-K，结果回取前K个作为检测结果

3、跨编译器TOP-1准确率较低

二进制代码比对研究现状

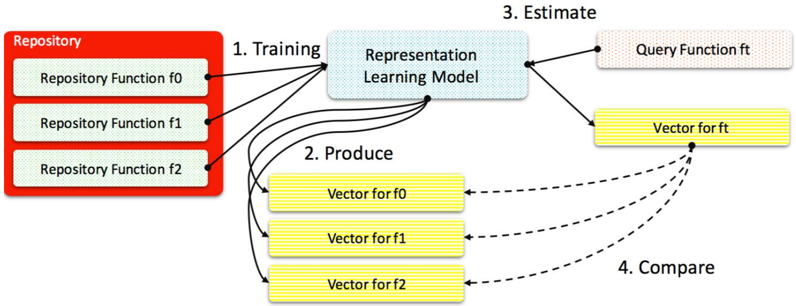
- Asm2Vec [S&P’19]：

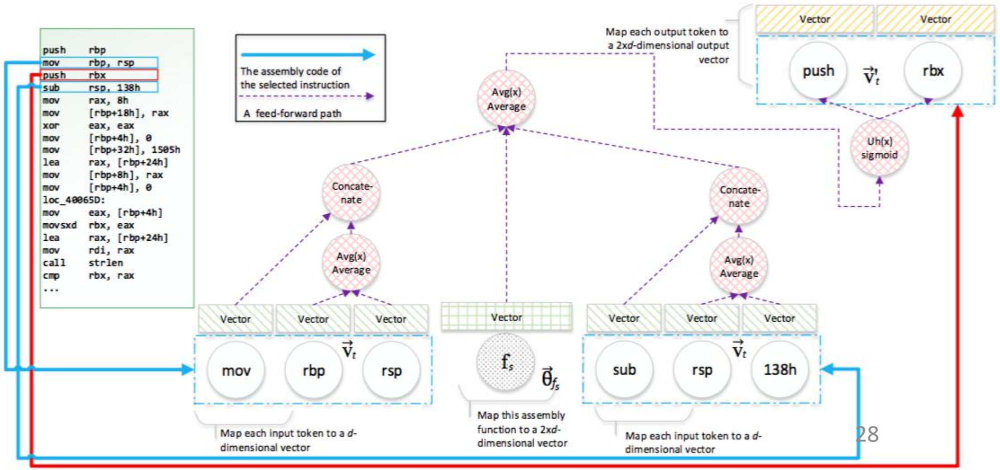
不可跨平台

基于自然语言处理

目标：使用语义表示实现抗代码混淆和编译优化的二进制同源检测

方法：利用PV-DM模型为每个函数生成语义向量，使用余弦距离判断同源相似度





二进制代码比对研究现状

- Asm2Vec [S&P’19]:

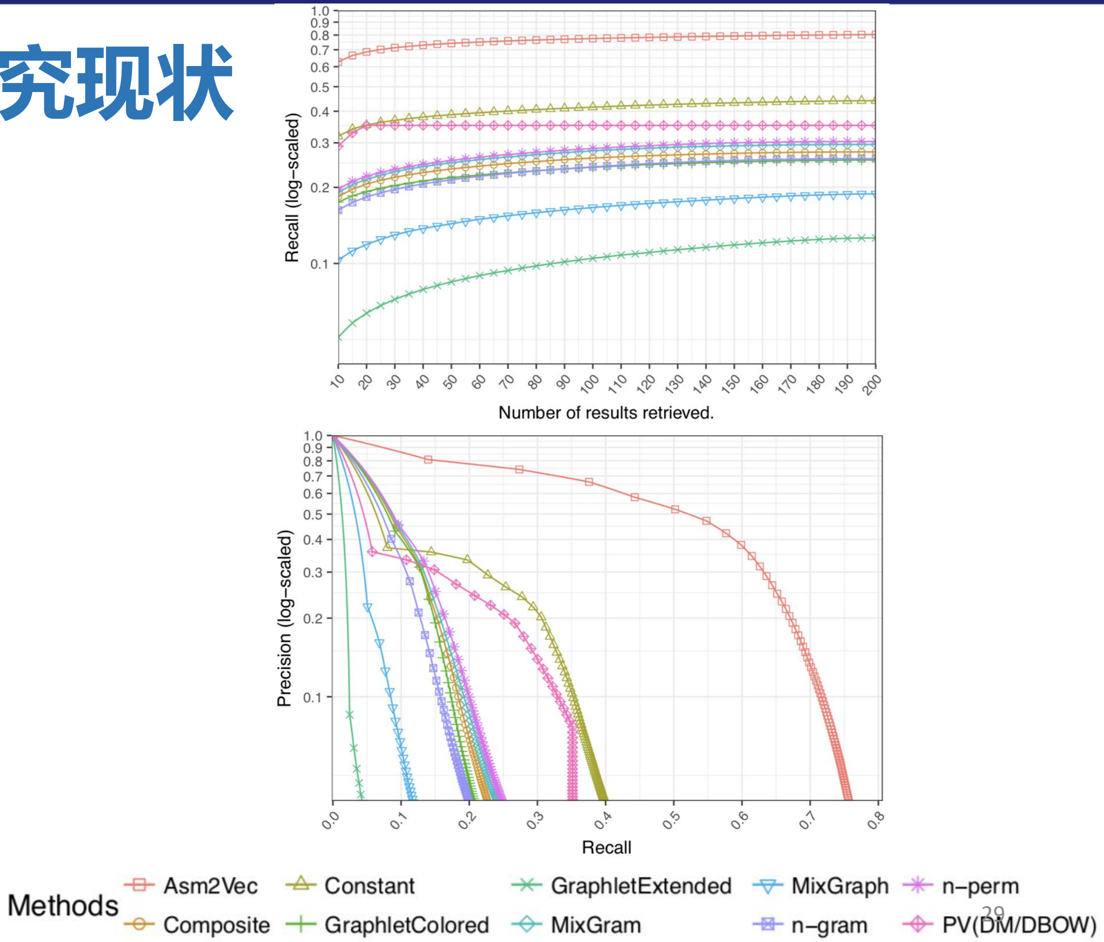
不可跨平台

基于自然语言处理

实验：与现有静态工具比较

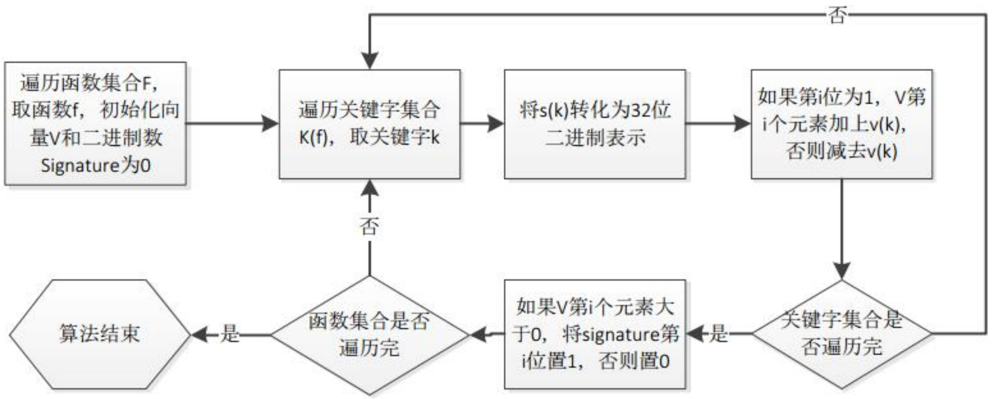
意义：证明了自然语言处理技术在二进制代码比对的有效性

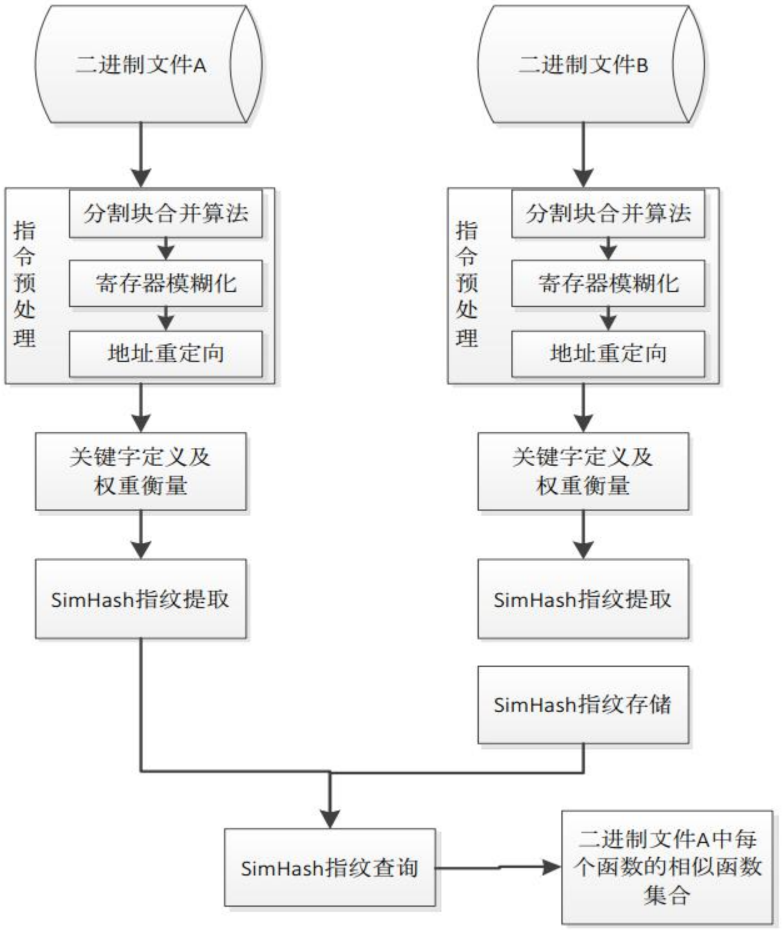
不足：局限于单一平台，无法进行跨平台的二进制代码比对



我们团队的工作

- 基于LLVM IR的跨架构函数比较算法
  - 使用RetDec工具将二进制文件转化为LLVM IR。
  - 使用改编的SimHash方法将函数转化为向量形式并计算函数间相似度





实验结果

- 准确性

跨指令架构的对比算法

场景：
相同版本的源码，
面向相同的架构，
使用不同的编译
选项进行编译

查询二进制	目标二进制	相似度准确率	优化后准确率
libssl.so.1.0.0 1.0.1f -O3 x64	libssl.so.1.0.0 1.0.1f -O2 x64	96.89%	97.63%
libssl.so.1.0.0 1.0.1f -O3 x64	libssl.so.1.0.0 1.0.1f -O1 x64	93.05%	94.31%
libssl.so.1.0.0 1.0.1f -O3 x64	libssl.so.1.0.0 1.0.1f -O0 x64	85.14%	87.66%
libssl.so.1.0.0 1.0.1f -O2 x64	libssl.so.1.0.0 1.0.1f -O1 x64	95.23%	96.26%
libssl.so.1.0.0 1.0.1f -O2 x64	libssl.so.1.0.0 1.0.1f -O0 x64	86.94%	89.44%
libssl.so.1.0.0 1.0.1f -O1 x64	libssl.so.1.0.0 1.0.1f -O0 x64	89.68%	92.04%
minizip 1.1.1 -O3 ARM	minizip 1.1.1 -O1 ARM	84.42%	84.42%
minizip 1.1.1 -O3 MIPS	minizip 1.1.1 -O2 MIPS	90.00%	91.67%

实验结果

• 准确性

跨指令架构的对比算法

场景：
相同版本的源码，面向不同的架构，使用相同的编译选项进行编译

查询二进制	目标二进制	相似度准确率	优化后准确率
libssl.so.1.0.0 1.0.1f -O3 x64	libssl.so.1.0.0 1.0.1f -O3 ARM	80.34%	85.37%
libssl.so.1.0.0 1.0.1f -O1 x64	libssl.so.1.0.0 1.0.1f -O1 ARM	81.43%	86.81%
libssl.so.1.0.0 1.0.1f -O2 MIPS	libssl.so.1.0.0 1.0.1f -O2 x64	61.83%	67.34%
libssl.so.1.0.0 1.0.1f -O3 MIPS	libssl.so.1.0.0 1.0.1f -O3 ARM	60.19%	67.22%
minizip 1.1.1 -O3 x64	minizip 1.1.1 -O3 ARM	87.01%	89.61%
minizip 1.1.1 -O1 MIPS	minizip 1.1.1 -O1 x64	68.18%	78.79%
minizip 1.1.1 -O2 MIPS	minizip 1.1.1 -O2 ARM	83.82%	88.24%

实验结果

- 准确性

跨指令架构的对比算法

场景：
相同版本的源码，
面向不同的架构，
使用不同的编译
选项进行编译

查询二进制	目标二进制	相似度准确率	优化后准确率
libssl.so.1.0.0 1.0.1f -O3 x64	libssl.so.1.0.0 1.0.1f -O1 ARM	77.38%	82.63%
libssl.so.1.0.0 1.0.1f -O1 x64	libssl.so.1.0.0 1.0.1f -O3 ARM	79.29%	85.26%
libssl.so.1.0.0 1.0.1f -O2 MIPS	libssl.so.1.0.0 1.0.1f -O3 ARM	58.33%	65.19%
libssl.so.1.0.0 1.0.1f -O3 MIPS	libssl.so.1.0.0 1.0.1f -O1 x64	60.33%	66.12%
minizip 1.1.1 -O3 ARM	minizip 1.1.1 -O2 x64	75.32%	79.22%
minizip 1.1.1 -O3 MIPS	minizip 1.1.1 -O2 ARM	85.00%	88.33%
minizip 1.1.1 -O2 MIPS	minizip 1.1.1 -O3 x64	54.41%	57.35%

实验结果

- 准确性

跨指令架构的对比算法

场景：
不同版本的源码，
面向相同的架构，
使用相同的编译
选项进行编译

查询二进制	目标二进制	相似度准确率	优化后准确率
libssl.so.1.0.0 1.0.1f -O3 x64	libssl.so.1.0.0 1.0.1g -O3 x64	99.77%	99.92%
libssl.so.1.0.0 1.0.1f -O3 ARM	libssl.so.1.0.0 1.0.1g -O3 ARM	98.74%	98.96%
libssl.so.1.0.0 1.0.1f -O3 MIPS	libssl.so.1.0.0 1.0.1g -O3 MIPS	95.87%	97.80%
libssl.so.1.0.0 1.0.1f -O3 x86	libssl.so.1.0.0 1.0.1g -O3 x86	97.34%	98.06%
minizip 1.1.0 -O3 x64	minizip 1.1.1 -O3 x64	81.28%	87.70%
minizip 1.1.0 -O1 ARM	minizip 1.1.1 -O1 ARM	95.24%	97.62
minizip 1.1.0 -O2 MIPS	minizip 1.1.1 -O2 MIPS	97.05%	98.52%

实验结果

- 准确性

跨指令架构的对比算法

场景：
不同版本的源码，
面向相同的架构，
使用不同的编译
选项进行编译

查询二进制	目标二进制	相似度准确率	优化后准确率
libssl.so.1.0.0 1.0.1f -O3 x64	libssl.so.1.0.0 1.0.1g -O2 x64	96.82%	97.56%
libssl.so.1.0.0 1.0.1f -O3 ARM	libssl.so.1.0.0 1.0.1g -O1 ARM	93.11%	94.15%
libssl.so.1.0.0 1.0.1f -O3 MIPS	libssl.so.1.0.0 1.0.1g -O0 MIPS	63.36%	69.83%
libssl.so.1.0.0 1.0.1f -O3 x86	libssl.so.1.0.0 1.0.1g -O2 x86	94.74%	95.46%
minizip 1.1.0 -O3 x64	minizip 1.1.1 -O2 x64	67.38%	75.40%
minizip 1.1.0 -O3 ARM	minizip 1.1.1 -O1 ARM	84.42%	84.42%
minizip 1.1.0 -O2 MIPS	minizip 1.1.1 -O1 MIPS	86.76%	88.24%

实验结果

- 准确性

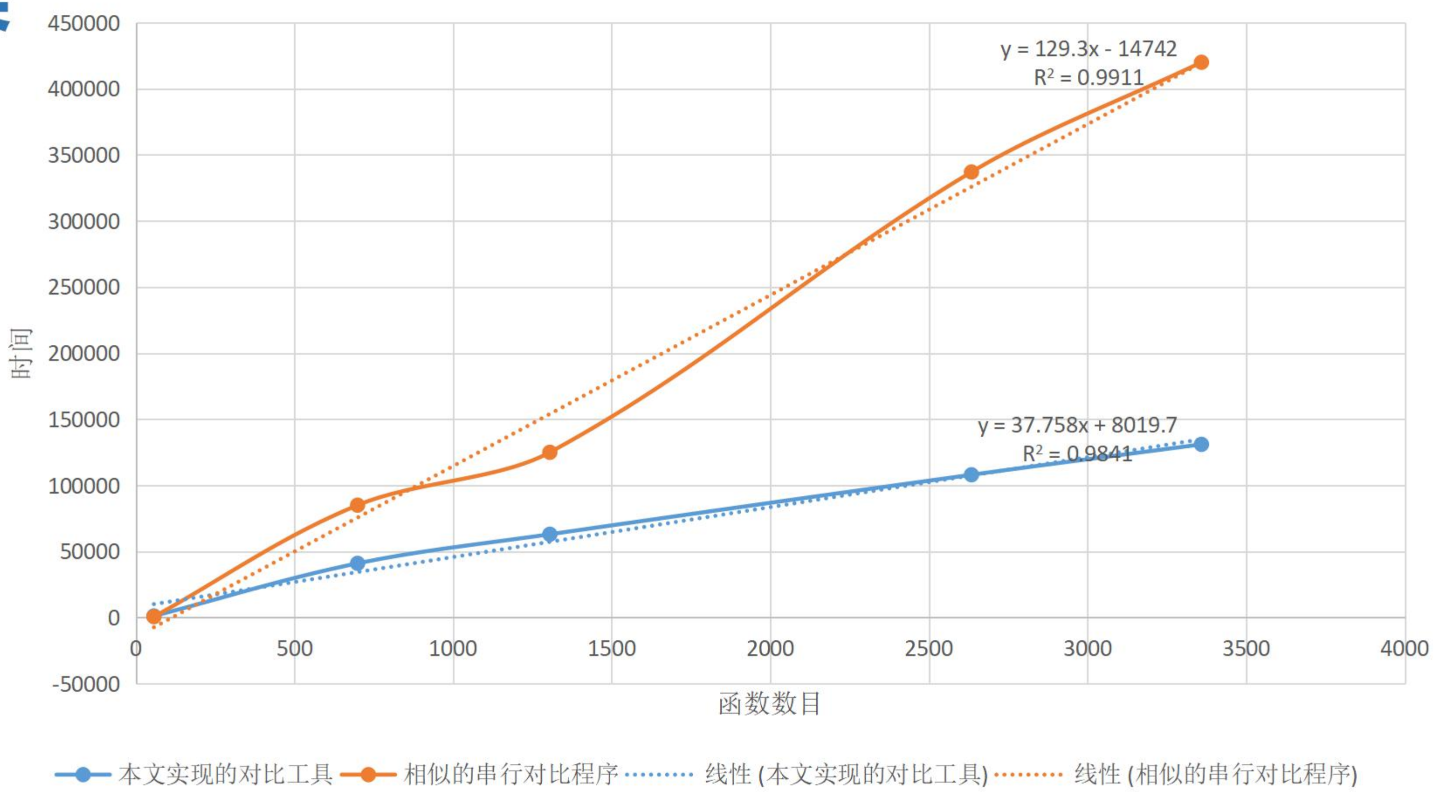
跨指令架构的对比算法

场景：
不同版本的源码，
面向不同的架构，
使用不同的编译
选项进行编译

查询二进制	目标二进制	相似度准确率	优化后准确率
libssl.so.1.0.0 1.0.1f -O3 x64	libssl.so.1.0.0 1.0.1g -O1 ARM	77.31%	82.71%
libssl.so.1.0.0 1.0.1f -O3 ARM	libssl.so.1.0.0 1.0.1g -O1 x86	82.22%	85.56%
libssl.so.1.0.0 1.0.1f -O3 MIPS	libssl.so.1.0.0 1.0.1g -O1 ARM	57.16%	64.05%
libssl.so.1.0.0 1.0.1f -O2 MIPS	libssl.so.1.0.0 1.0.1g -O3 x64	58.60%	64.65%
minizip 1.1.0 -O3 MIPS	minizip 1.1.1 -O1 x64	61.67%	63.33%
minizip 1.1.0 -O2 MIPS	minizip 1.1.1 -O1 ARM	76.47%	79.41%
minizip 1.1.0 -O3 ARM	minizip 1.1.1 -O2 x64	74.03%	79.22%

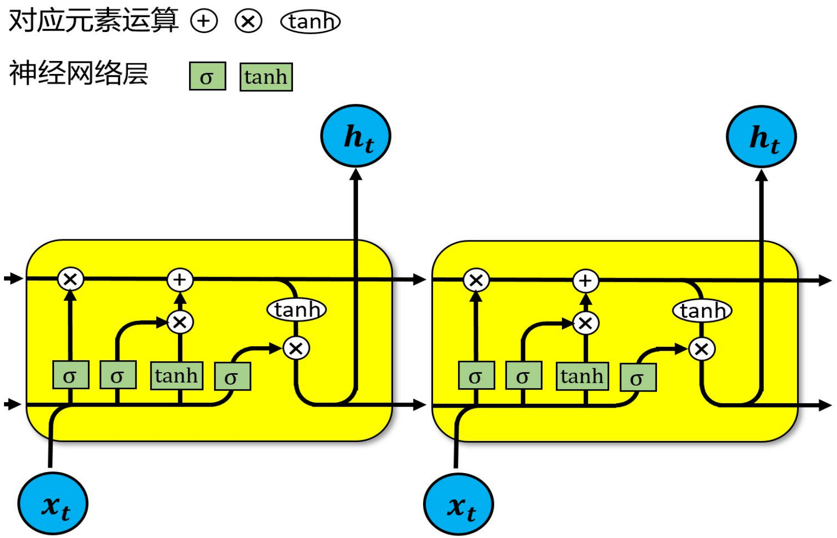
实验结果

- 性能


y = 129.3x - 14742
R² = 0.9911
y = 37.758x + 8019.7
R² = 0.9841
时间
函数数目
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
-50000
0
500
1000
1500
2000
2500
3000
3500
4000
本文实现的对比工具
相似的串行对比程序
线性 (本文实现的对比工具)
线性 (相似的串行对比程序)

我们团队的工作

- 基于LSTM的孪生网络的函数比较算法
  - 使用自然语言处理的词向量模型将指令映射到语义向量。
  - 使用LSTM循环神经网络，将基本块的指令语义向量整合成基本块的语义向量。



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

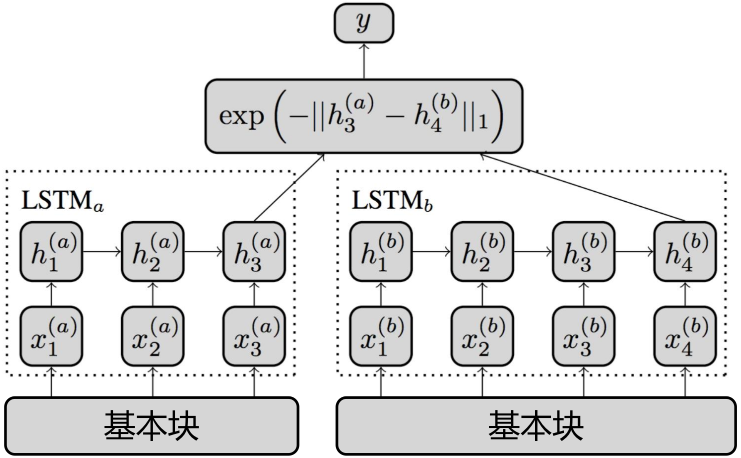
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

我们团队的工作

- 基于LSTM的孪生网络的函数比较算法
  - 为每一个基本块生成其语义向量之后，使用孪生网络实现跨平台的两个函数基本块的同源比较。
  - 根据计算的两个函数之间基本块同源映射的关系，计算两个跨平台函数的同源相似程度。
  - SimFunc(A,B)=SimBB(A,B)/AllBB(A,B)

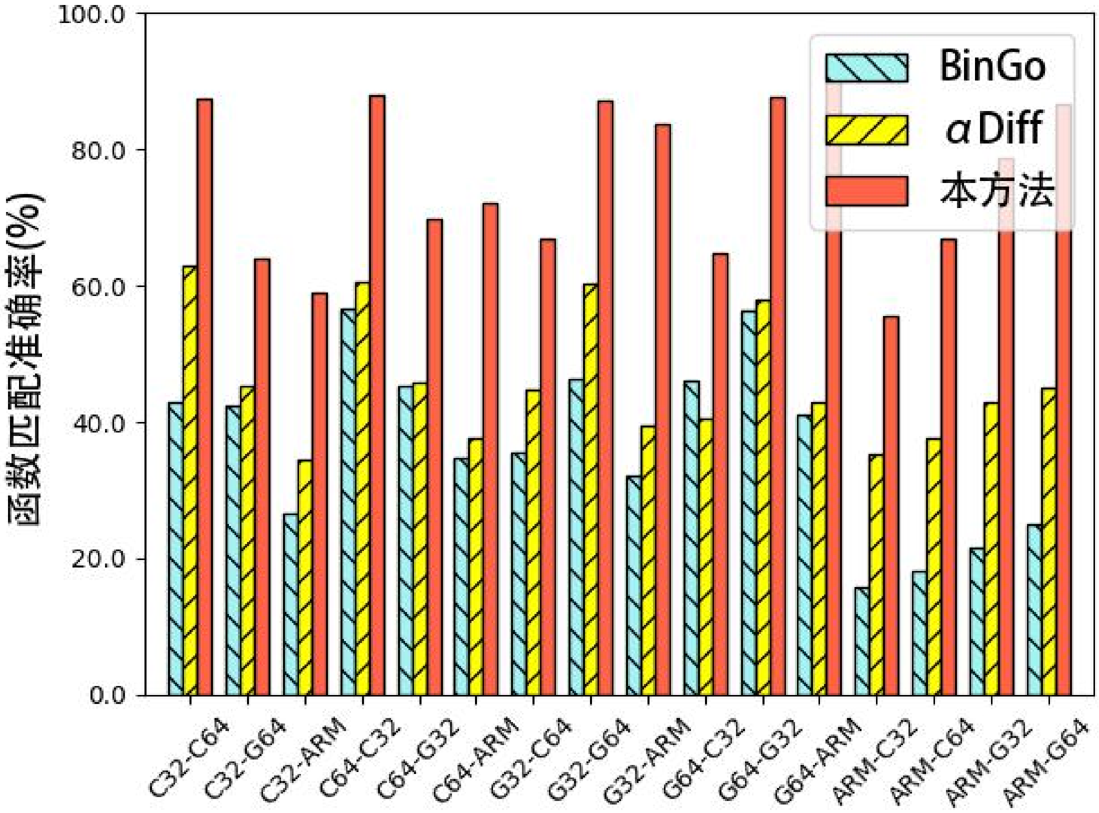


实验结果

- 准确性

跨指令架构的对比算法

比较：
相同版本的源码，
面向不同的架构，
使用不同的编译
器进行编译



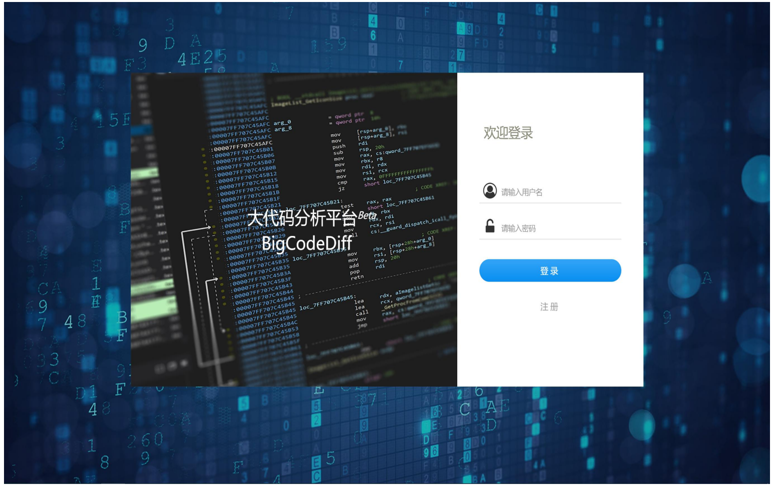
提 纲

- 二进制代码比对分析的作用
- 研究现状与我们的研究
- 二进制代码分析云平台BigCodeDiff

BigCodeDiff：并行化、算法可扩展的二进制代码比对分析云平台

系统主要功能特性

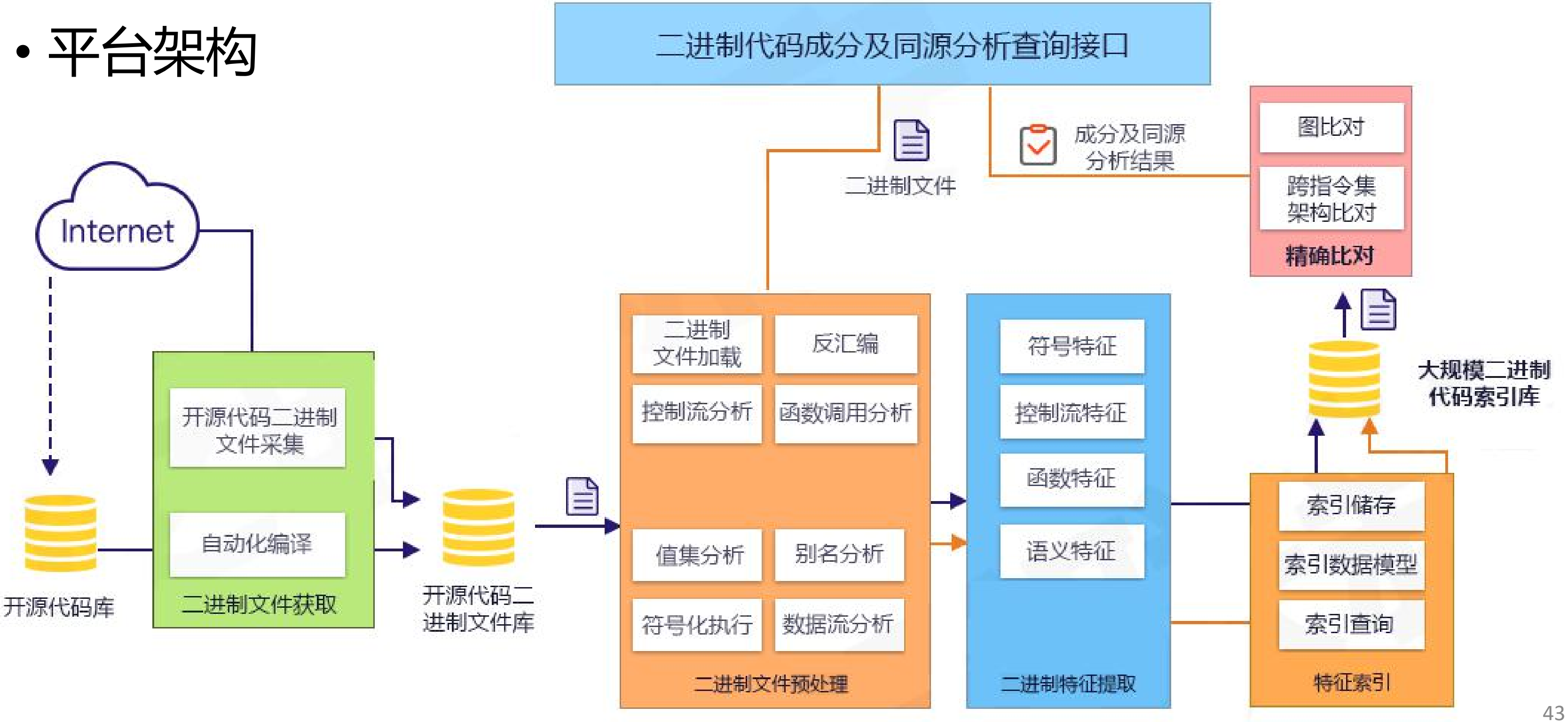
- 细粒度
  - 函数级粒度的比对，支持x86、x64、ARM、MIPS等多种指令集代码。
- 高性能
  - 采用分布式并行计算、图数据库、高效索引等机制，实现对海量代码进行高性能分析。
- 可扩展
  - 以插件方式集成机器学习等多种独创和学术界最新算法，从精度/性能的平衡、是否支持跨指令集、是否抗混淆等方面满足不同应用场景。
- 界面友好
  - 具有丰富的图形化表示，在指令级、控制流图、函数调用图等层次提供相似度的直观显示。
- 支持多文件类型
  - 支持对pe文件、elf文件、idb文件、固件等类型文件进行比对分析。



集成算法：
基于基本块子图搜索的一对多检测算法
基于表示学习和语义向量运算的一对多检测算法
基于符号执行和抽象语法图搜索的一对多检测算法
基于启发式函数比对的一对一检测算法
基于中间语言LLVM IR特征的一对一检测算法

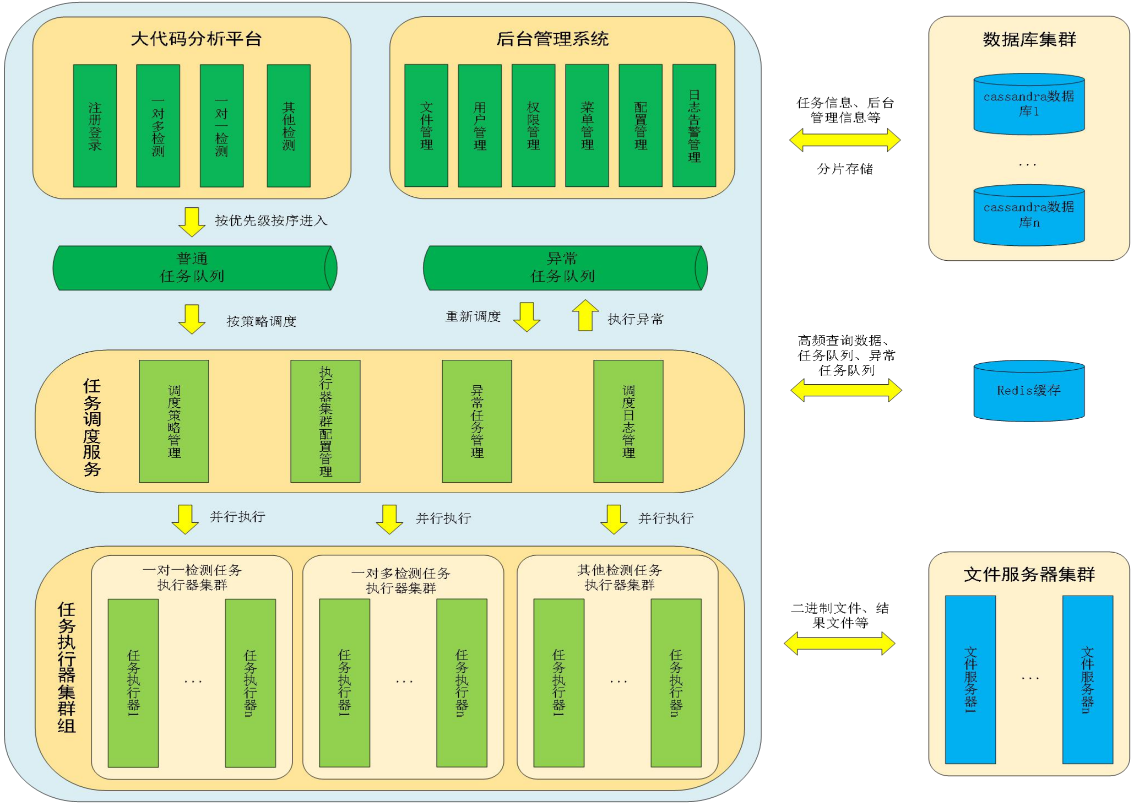
二进制代码分析云平台BigCodeDiff

- 平台架构



二进制代码分析云平台BigCodeDiff

- 平台架构

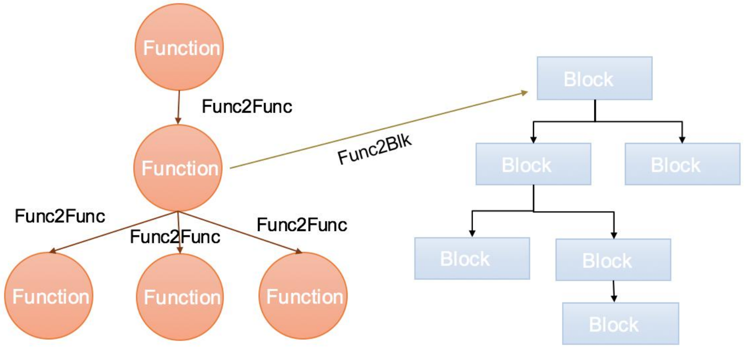


数据存储模型

- 图数据库采用JanusGraph

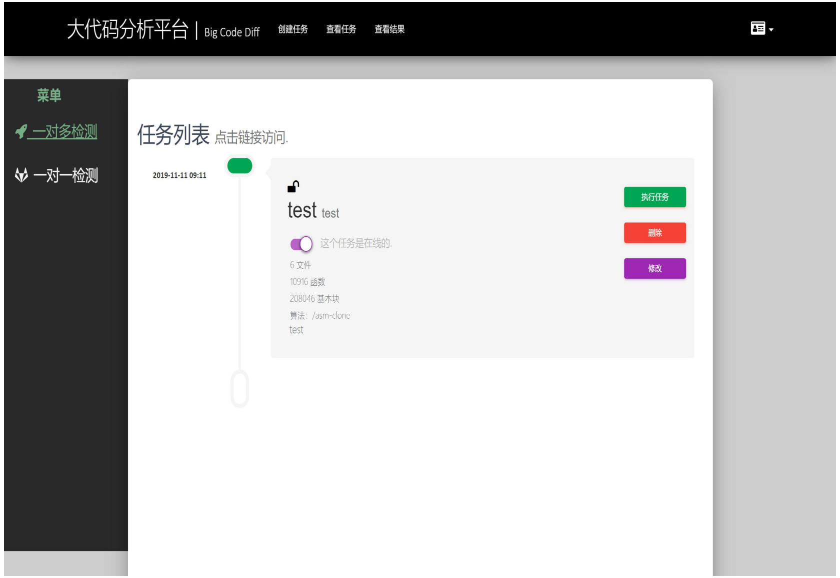
JanusGraph

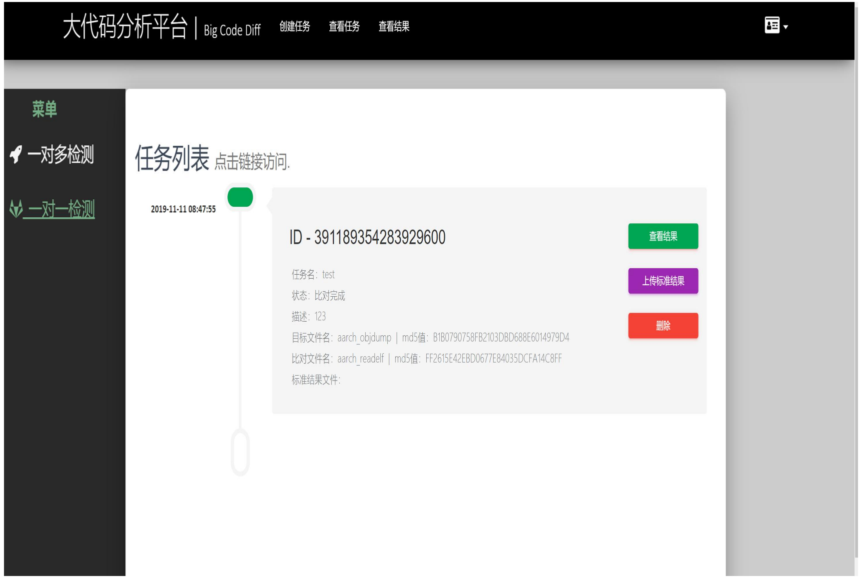
- 图数据模型：
  - 每一个Janusgraph图形都有一个由边标签、属性和顶点标签组成的模式。Janusgraph模式可以显式或隐式定义，为了提高应用的健壮性，提高协同开发的效率，采用显示定义的方式。
  - 在二进制文件中有两种图：函数调用图和控制流图。



平台使用

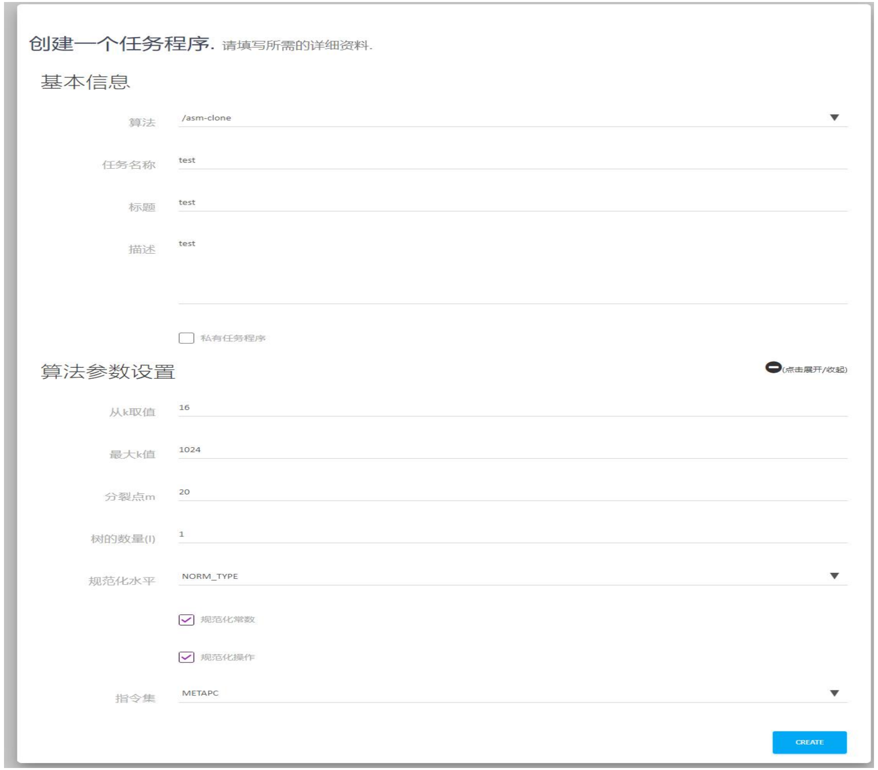
- 平台域名： http://www.bigcodediff.com





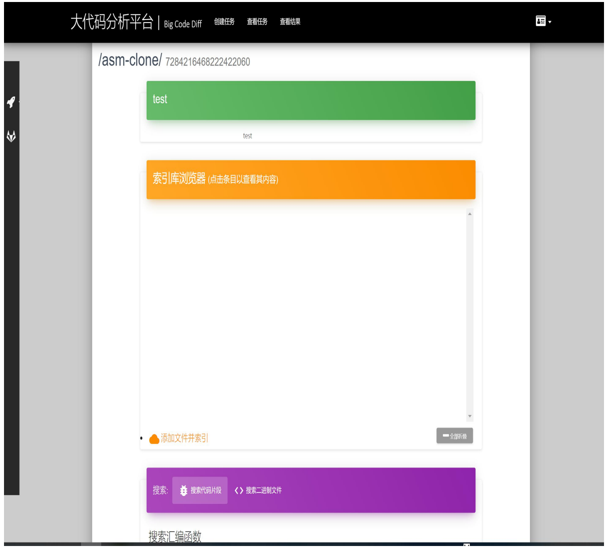
平台使用

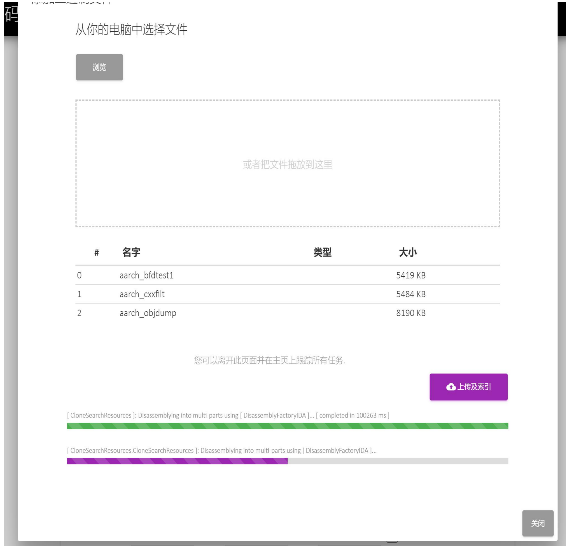
- 创建一对多任务

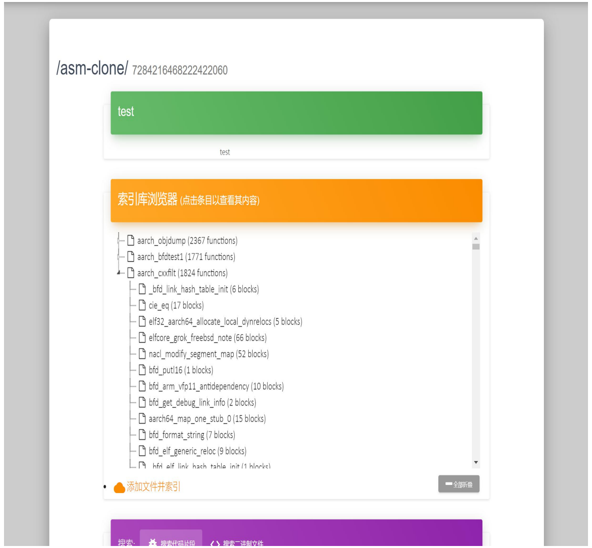


平台使用

- 建立索引库

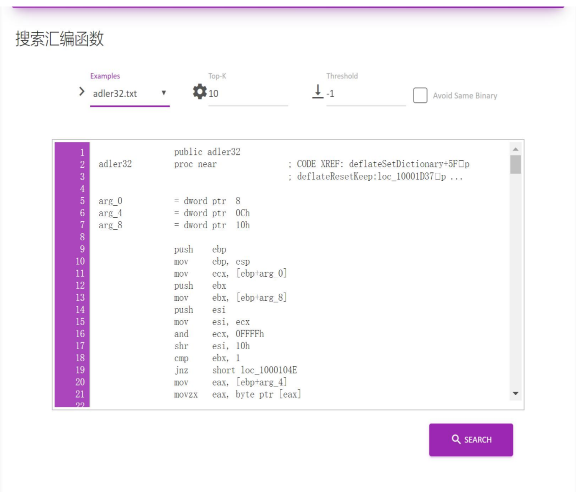

大代码分析平台 | Big Code Diff
创建任务
查看任务
查看结果
/asm-clone/ 7284216468222422060
test
test
索引库浏览器 (点击条目以查看其内容)
添加文件并索引
搜索:
搜索代码片段
搜索二进制文件
搜索汇编函数

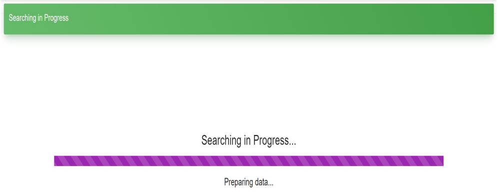

从你的电脑中选择文件
浏览
或者把文件拖放到这里
#
名字
类型
大小
0
aarch_bfdtest1
5419 KB
1
aarch_cxxfilt
5484 KB
2
aarch_objdump
8190 KB
您可以离开此页面并在主页上跟踪所有任务.
上传及索引
关闭

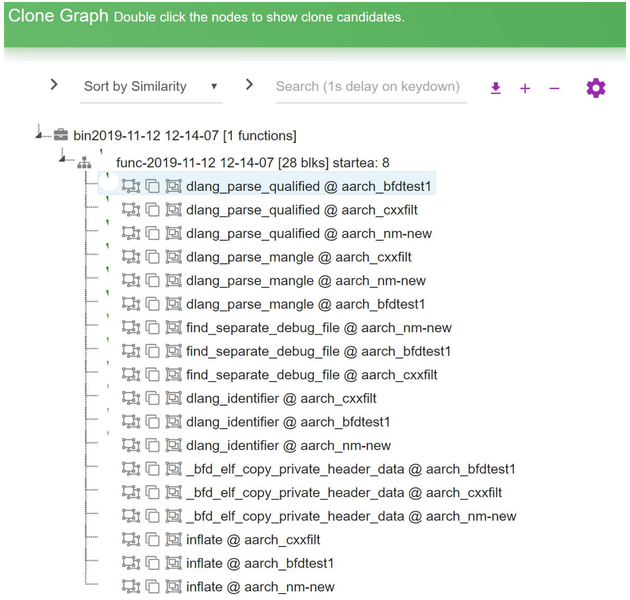

/asm-clone/ 7284216468222422060
test
test
索引库浏览器 (点击条目以查看其内容)
aarch_objdump (2367 functions)
aarch_bfdtest1 (1771 functions)
aarch_cxxfilt (1824 functions)
_bfd_link_hash_table_init (6 blocks)
cie_eq (17 blocks)
elf32_aarch64_allocate_local_dynrelocs (5 blocks)
elfcore_grok_freebsd_note (66 blocks)
nacl_modify_segment_map (52 blocks)
bfd_putl16 (1 blocks)
bfd_arm_vfp11_antidependency (10 blocks)
bfd_get_debug_link_info (2 blocks)
aarch64_map_one_stub_0 (15 blocks)
bfd_format_string (7 blocks)
bfd_elf_generic_reloc (9 blocks)
添加文件并索引

平台使用

- 搜索汇编函数

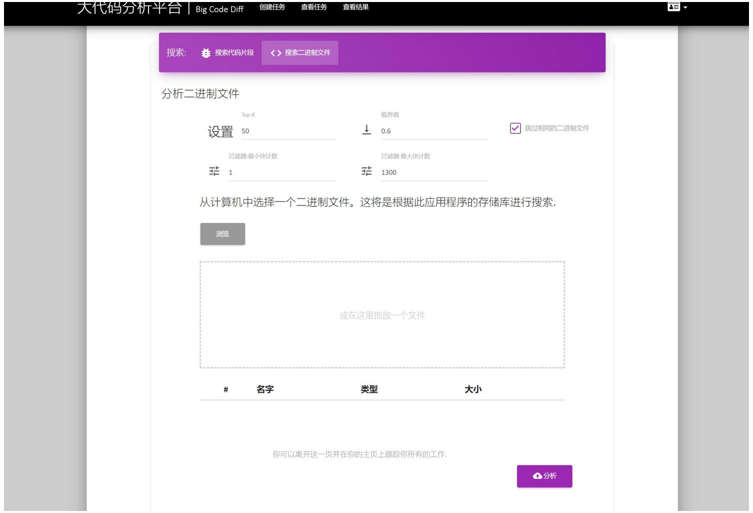

搜索汇编函数
Examples
adler32.txt
Top-K
10
Threshold
-1
Avoid Same Binary
public adler32
adler32 proc near ; CODE XREF: deflateSetDictionary+5F□p
; deflateResetKeep:loc_10001D37□p ...
arg_0 = dword ptr 8
arg_4 = dword ptr 0Ch
arg_8 = dword ptr 10h
push ebp
mov ebp, esp
mov ecx, [ebp+arg_0]
push ebx
mov ebx, [ebp+arg_8]
push esi
mov esi, ecx
and ecx, 0FFFFh
shr esi, 10h
cmp ebx, 1
jnz short loc_1000104E
mov eax, [ebp+arg_4]
movzx eax, byte ptr [eax]
SEARCH

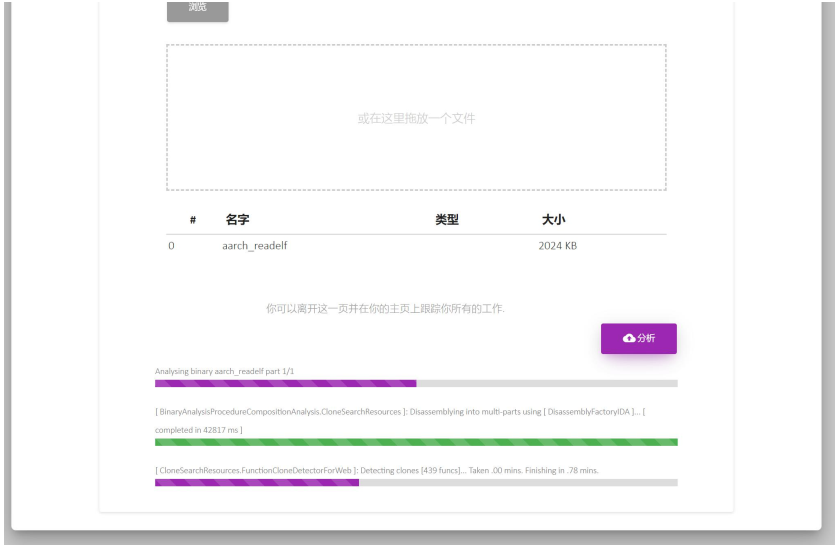

Searching in Progress
Searching in Progress...
Preparing data...


Clone Graph Double click the nodes to show clone candidates.
Sort by Similarity
Search (1s delay on keydown)
bin2019-11-12 12-14-07 [1 functions]
func-2019-11-12 12-14-07 [28 blks] startea: 8
dlang_parse_qualified @ aarch_bfdtest1
dlang_parse_qualified @ aarch_cxxfilt
dlang_parse_qualified @ aarch_nm-new
dlang_parse_mangle @ aarch_cxxfilt
dlang_parse_mangle @ aarch_nm-new
dlang_parse_mangle @ aarch_bfdtest1
find_separate_debug_file @ aarch_nm-new
find_separate_debug_file @ aarch_bfdtest1
find_separate_debug_file @ aarch_cxxfilt
dlang_identifier @ aarch_cxxfilt
dlang_identifier @ aarch_bfdtest1
dlang_identifier @ aarch_nm-new
_bfd_elf_copy_private_header_data @ aarch_bfdtest1
_bfd_elf_copy_private_header_data @ aarch_cxxfilt
_bfd_elf_copy_private_header_data @ aarch_nm-new
inflate @ aarch_cxxfilt
inflate @ aarch_bfdtest1
inflate @ aarch_nm-new

平台使用

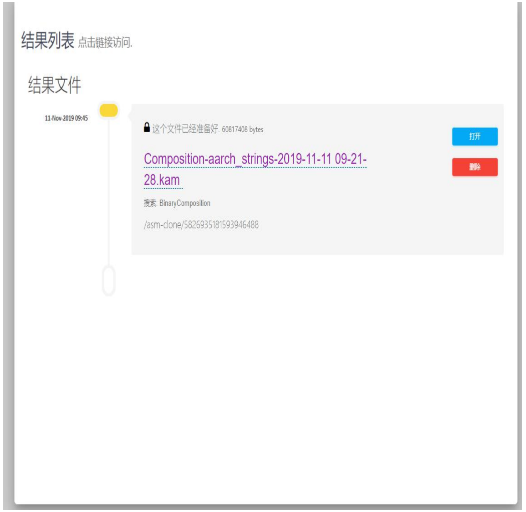
- 搜索二进制文件

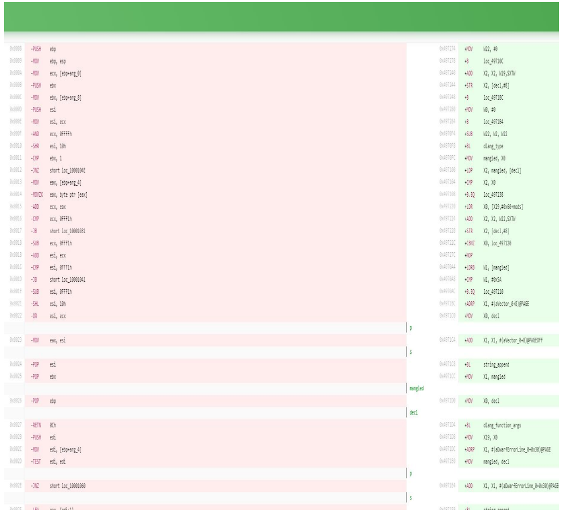


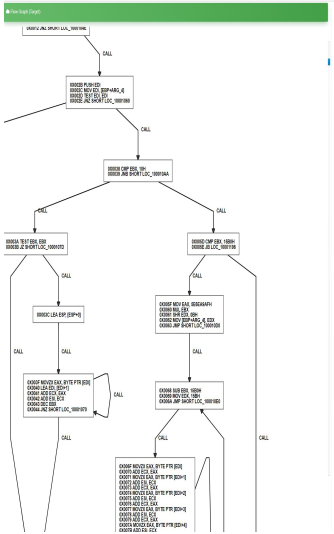


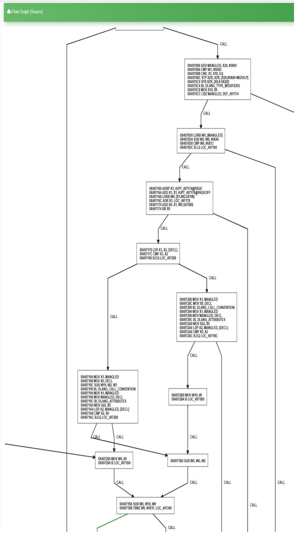
平台使用

- 搜索比对结果


结果列表 点击链接访问.
结果文件
11-Nov-2019 09:45
这个文件已经准备好. 60817408 bytes
打开
删除
Composition-aarch_strings-2019-11-11 09-21-28.kam
搜索: BinaryComposition
/asm-clone/5826935181593946488

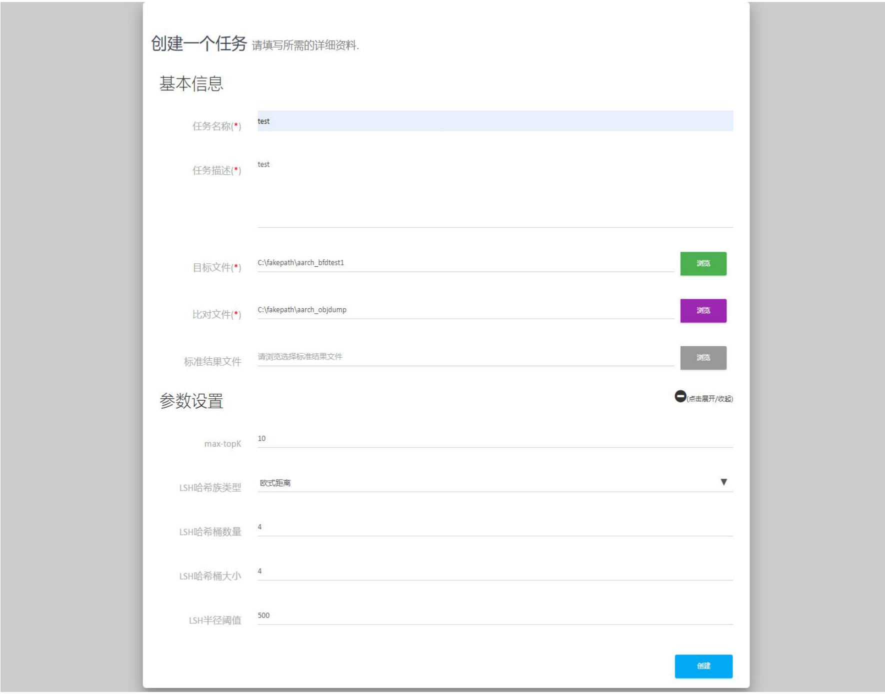






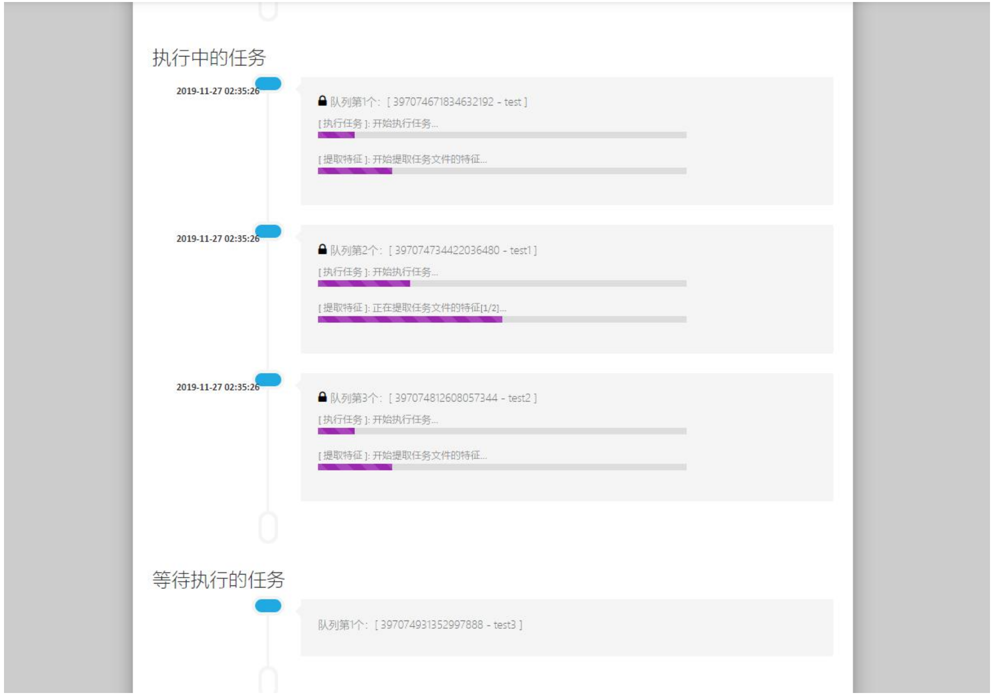
平台使用

- 创建一对一任务



平台使用

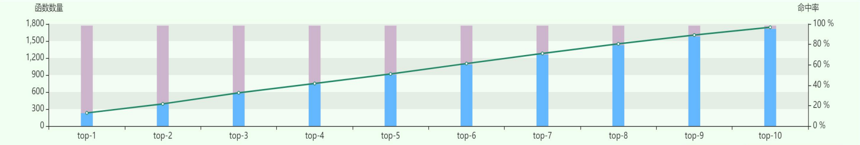
- 执行一对一任务



平台使用

- 一对一比对结果

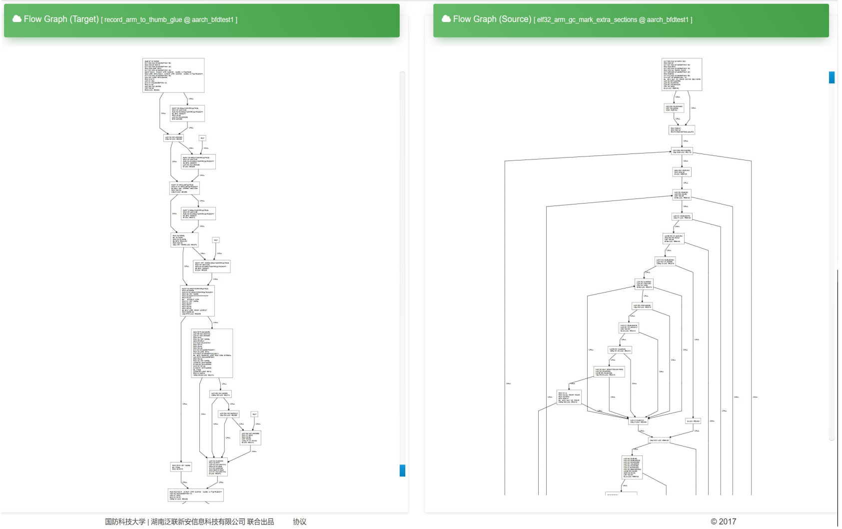
比对结果



搜索目标函数　查询　按函数名评估命中率

函数名【文件名】	相似度	函数内部流图	函数调用关系图	是否命中	备注
record_arm_to_thumb_glue @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null
elf_slurp_reloc_table_from_section @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null
bfd_get_arch @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null
_bfd_elf_check_kept_section @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null
ext32l_r_offset @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null
_bfd_elf_strtab_init @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null
inflateCodesUsed @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null
_bfd_elf_get_special_section @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null
bfd_elf64_core_file_failing_command @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null
elf32_arm_gc_mark_extra_sections @ aarch_bfdtest1 [0 blks]	100%	-	-	✔	null

共 1770 条，每页 10 条，本页 10 条，共 177 页　首页 << 1 2 3 4 5 6 7 8 9 10 >> 尾页 跳转



XDef 2019

对话 · 交流 · 合作　　前沿 · 实用 · 人才

Thanks

谢谢关注！